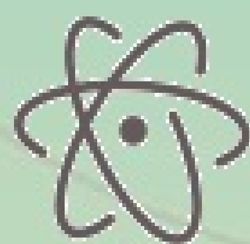


ATOM



目錄

Atom飞行手册翻译

翻译自 Atom Flight Manual。

- 在线阅读
- PDF下载
- EPUB下载
- MOBI下载

赞助我



协议

新手入门

为什么选择Atom？

来源：https://github.com/atom-china/manual

这个世界上有那么多种编辑器，为什么你要花时间学习和使用 Atom 呢？

虽然 Sublime 和 TextMate 之类的编辑器已经非常好用了，但它们仅提供了很有限的拓展性。而在另一个极端，Emacs 和 Vim 提供了灵活的拓展性，但它们并不是很友好，需要使用专用的编程语言来配置和拓展。

我们觉得我们可以做得更好。我们的目标是在保证易用性的同时提供充分的可拓展性（hackability）：这个编辑器会受到第一天学习编程的新生欢迎，而且当他们成长为编程专家时也难以割舍。

当我们使用 Atom 来开发 Atom 的时候，随着它的逐渐完善，我们愈发觉得已经离不开它了。从表面上来看，Atom 是一个能满足你的期待的，现代化的桌面文本编辑器，而在表面之下，这是一个值得你去一同完善的系统。

Atom 的核心

Web 技术虽然有其缺陷，但经过二十年的发展，Web 已经逐渐成长为了一个强大的具有活力的平台。所以当我们计划写一个自用的可拓展的文本编辑器时，Web 技术显然是一个好的选择，但首先我们需要摆脱来自 Web 的限制。

混合本地代码与 Web 技术

Web 浏览器很适合用来浏览网页，但写代码是一种需要可靠的工具的专业活动。更重要的是，浏览器出于安全的考虑，严格限制了对本地系统的访问，但对一个文本编辑器而言，不能向本地系统写入文件是不可接受的。

因此，我们没有把 Atom 构建为一个传统的 Web 应用，Atom 是一个专门被设计用作文本编辑器，而不是网页浏览器的 Chromium 定制版。Atom 的每一个窗口实际上都是一个本地渲染的网页。

所有来自 Node.js 可用的 API 在 Atom 窗口的 JavaScript 中同样可用，这种结合带来了一种独一无二的开发体验。

因为一切都是本地的，你不需要将静态资源打包、不需要关注脚本的异步加载，如果你希望加载一些代码。只需要在文件的最顶部 `require` 它即可，Node.js 的模块系统允许你将一个系统分割为小的、专注于某一功能的包。

JavaScript 与 C++ 的结合

与原生代码交互也很简单。例如，你基于 Oniguruma 正则引擎开发了一个用来提供对 TextMate 语法识别的支持。在浏览器里，你可能需要使用 NaCl 或 Esprima, 而在 Node 里这个过程变得非常简单。

在 Node.js 的 API 之外，我们还提供了一些 API 例如使用系统的对话框、使用菜单栏和右键菜单、操纵窗口尺寸等等。

Web 技术：最有趣的部分

另一个好消息就是当你为 Atom 编写代码时，这些代码一定会被允许在最新版本的 Chromium 中。这意味着你可以无视与浏览器兼容性有关的黑科技，使用全部的最新的 Web 功能。

例如，Atom 的工作区和窗格都是基于 flexbox 来进行布局的。这是一项刚刚出现的技术，从我们使用它之后也发生了很多变化，但不要紧，因为它工作得很好。

我们确信将 Atom 构建在 Web 技术之上是一个好的选择，因为整个行业都在推动着 Web 技术的发展。原生UI技术不断产生又不断淘汰，而 Web 是一个每年都变得更加强大和普及的标准。我们对于深入探索这一强大的技术感到无比兴奋。

一个开源的文本编辑器

GitHub 的目标是帮助大家构建更好的软件，而 Atom 则是实现这一目标的重要补充。Atom 是一项长期的投资，GitHub 会持续投入开发力量来推动它的发展。但我们也意识到不能让它受限于我们的能力，就像之所以 Emacs 和 Vim 在过去的三十年间被广泛使用，是因为只有开源，才能构建一个持久的、有活力的文本编辑器社区。

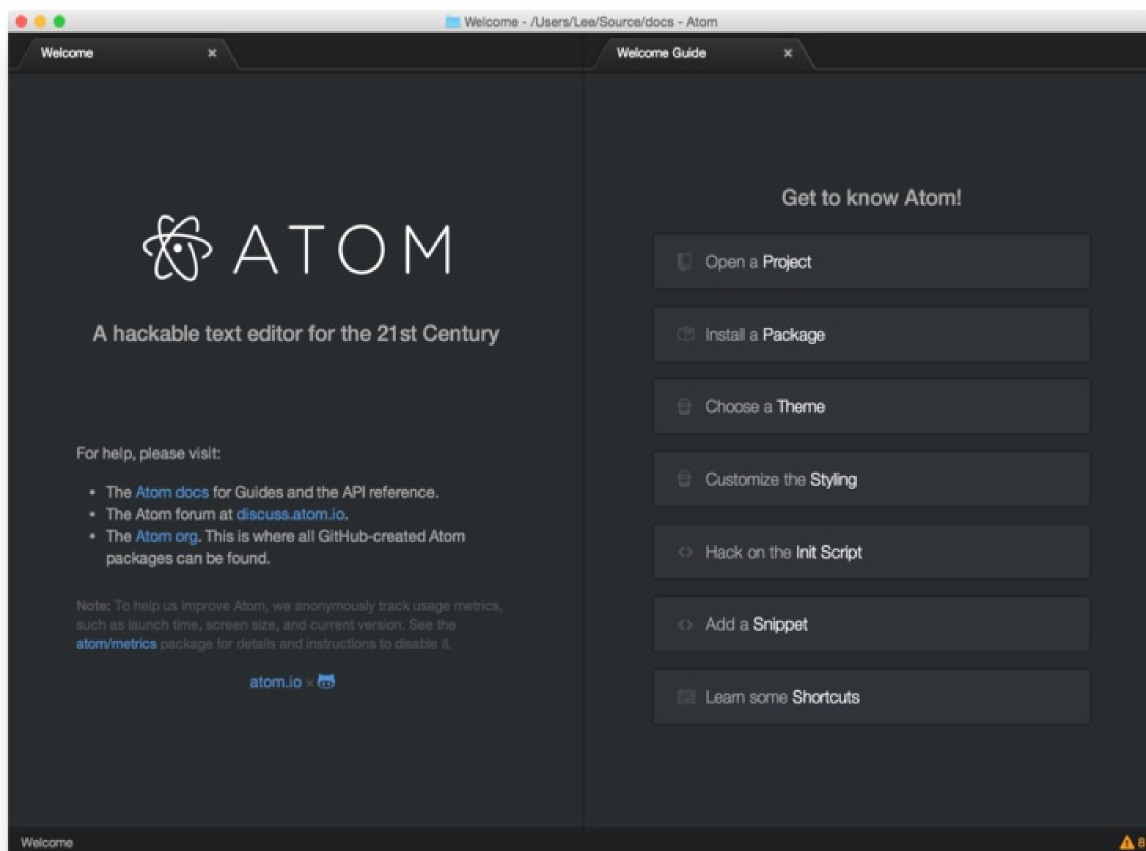
整个 Atom 编辑器都是免费且开源的，你可以在 https://github.com/atom 这个组织下找到它。

安装Atom

来源：https://github.com/atom-china/manual

当你安装好了 Atom 之后，让我们来认识一下它吧。

当你第一次打开 Atom 的时候，你会看到这样的一个窗口：



这是 Atom 的欢迎屏幕（welcome screen），它展示了一些不错的建议，帮助你了解 Atom.

基本术语

让我们先来了解一下接下来要用到的几个术语：

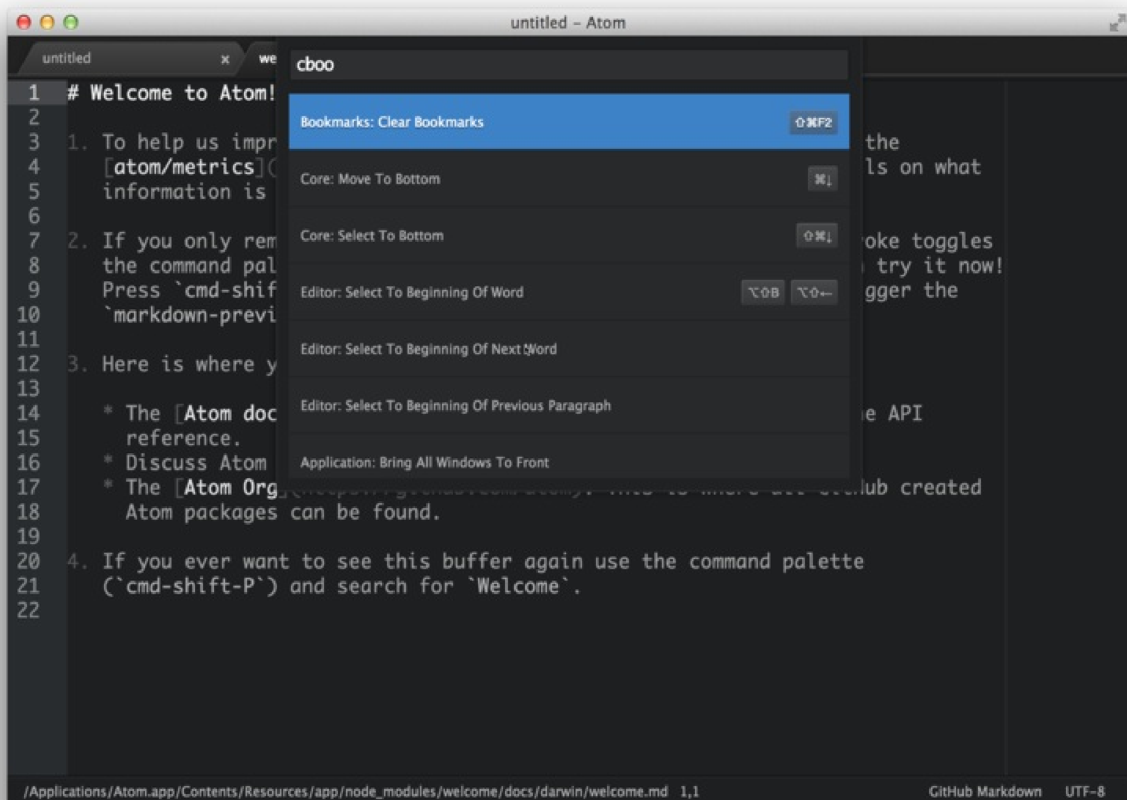
> 缓冲区（**Buffer**）代表了 Atom 中的一个文件的文本内容，它基本上相当于一个真正的文件，但它是被 Atom 维护在内存中的，如果你修改了它，在你保存之前，缓冲区的内容都不会被写入到硬盘里。
>
> 窗格（**Pane**）代表 Atom 中的一个可见区域。例如在欢迎屏幕上你可以看到四个窗格：用来切换文件的标签栏（tab bar），用来显示行号的边框（gutter），底部的状态栏（status bar），以及文本编辑器。

命令面板

当你按下 `cmd-shift-P` 并且当前焦点在一个窗格上的时候，命令面板就会弹出来。

在这个教程中我们会用类似 `cmd-shift-P` 的形式来运行命令，这是 Atom 在 Mac 上的默认快捷键，如果你在其他的平台上使用 Atom, 可能会稍有不同。如果某个快捷键无法工作，你可以通过命令面板来查找正确的快捷键。

在 Atom 中几乎所有的操作都通过这种搜索驱动的菜单来完成，你只需要按下 `cmd-shift-P` 来搜索命令，而不必在复杂的传统菜单栏间点来点去。

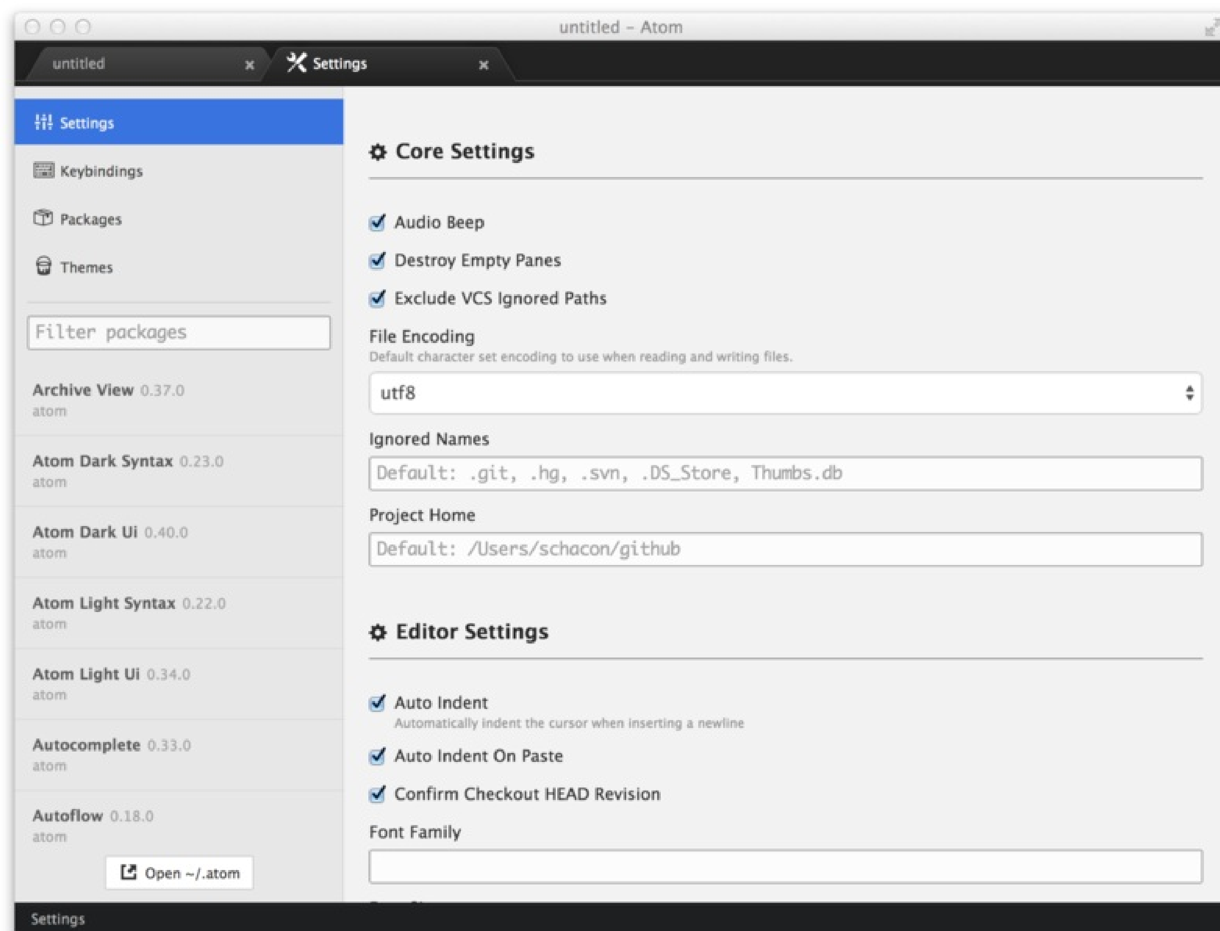


除了搜索数以千计的命令之外，命令面板上会显示每个命令对应的快捷键，这意味着你可以在使用这些命令的同时学习对应的快捷键，以便之后使用。

在本书的剩余部分我们会向你介绍一些命令，你可以在命令面板中搜索，或使用对应的快捷键。

偏好设置

Atom 有很多选项和偏好设置，你可以在设置界面修改它们。

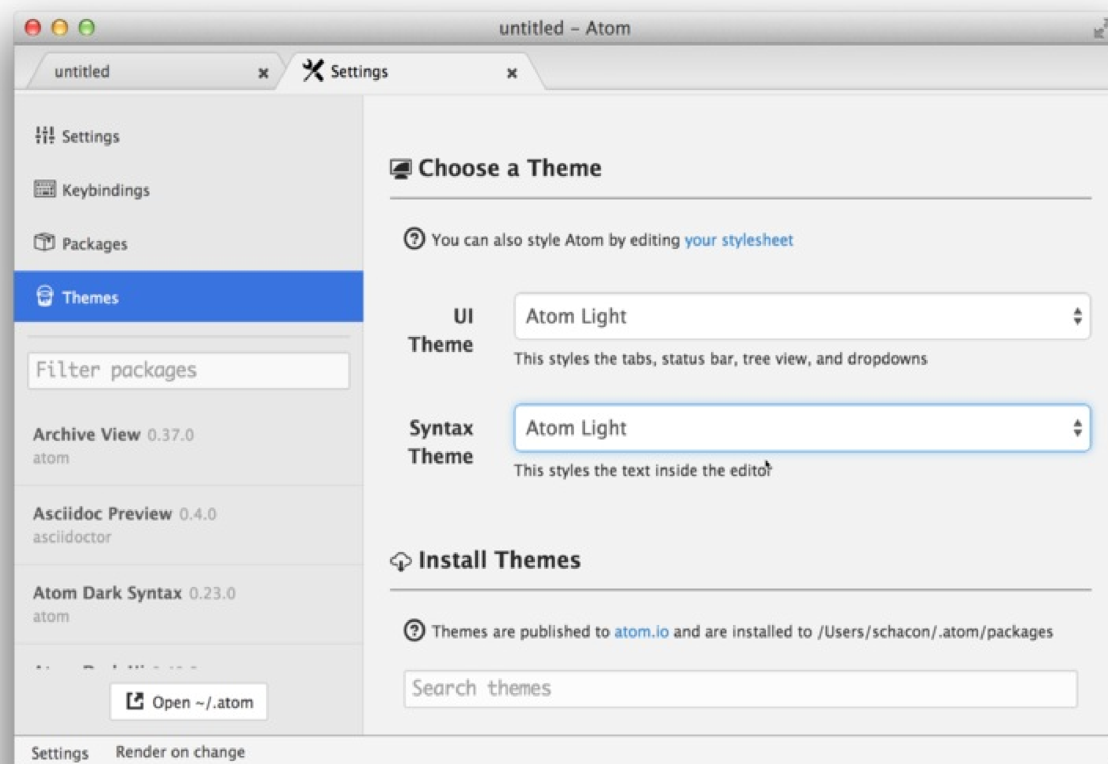


在设置界面中，你可以修改主题、修改文本折行的行为（wrapping）、字体大小、缩进宽度、滚动速度等选项。你也可以用这个界面安装新的插件和主题，我们在 Atom Packages 中介绍了这个话题。

你可以通过菜单栏中 Atom 下的 Preferences 菜单项打开设置界面。你也可以在命令面板中搜索 `settings-view:open` 或使用快捷键 `cmd-,` .

修改主题

你可以在设置界面中修改 Atom 的主题，Atom 内建了 4 个不同的 UI 主题，分别是亮色和暗色版本的名为 Atom 和 One 的主题。以及 8 个不同的语法着色主题。你可以通过点击左边栏的 Themes 选项卡来改变当前主题，或安装新的主题。



UI 主题会修改标签栏、左侧目录树（tree view）等 UI 元素的颜色；而语法着色主题修改编辑器中文字的语法高亮方案。你只需要简单地在下拉框中选择另一项，即可修改主题。

在 Atom.io 上有数十种主题供你选择，如果你想要一个独一无二的主题，我们也在 Basic Customization 中介绍了如何自定义主题，还在 Creating a Theme 中介绍了如何创建一个主题。

文本折行

你可以通过设置界面指定 Atom 处理空白和折行的策略。

- [ ] Show Indent Guide
- [ ] Show Invisibles
- [x] Show Line Numbers
- [x] Soft Tabs
- [x] Soft Wrap
- [ ] Soft Wrap At Preferred Line Length

Tab Length

2

当你启用了 Soft Tabs, Atom 将会在你按 `tab` 键时用空格来替代真正的制表符，Tab Length 则指定了一个制表符代表多少个空格，或者当 Soft Tabs 被禁用时多少个空格相当于一个制表符。

如果开启了 Soft Wrap 选项，Atom 会在一行中的文本超出屏幕显示范围时将其折为两行，如果禁用了这个选项，过长的行将简单地超出屏幕显示范围，你必须要横向移动滚动条才能看到剩余的部分。如果 Soft Wrap At Preferred Line Length 选项被开启，则总是会在 80 个字符处折行，你也可以设置一个自定义的长度来替换掉默认的 80 个字符。

在 Basic Customization 中我们会介绍如何为不同的文件类型（例如你希望在 Markdown 文件中折行，但不希望在代码中也是如此）设置不同的折行配置。

测试功能（**Beta Features**）

在 Atom 的开发过程中，偶尔会有一些新功能，但没有默认启用给所有用户。如果你愿意的话，你可以在设置界面中提前体验这些功能。

Use Hardware Acceleration
Disabling will improve editor font rendering but reduce scrolling performance.

Use Shadow DOM
Enable to test out themes and packages with the new shadow DOM before it ships by default.

Zoom Font When Ctrl Scrolling
Increase/decrease the editor font size when pressing the Ctrl key and scrolling the mouse up/down.

这对于插件开发者来说非常有用，开发者可以在一个功能被默认启用之前，测试他们维护的插件与新功能的兼容性。

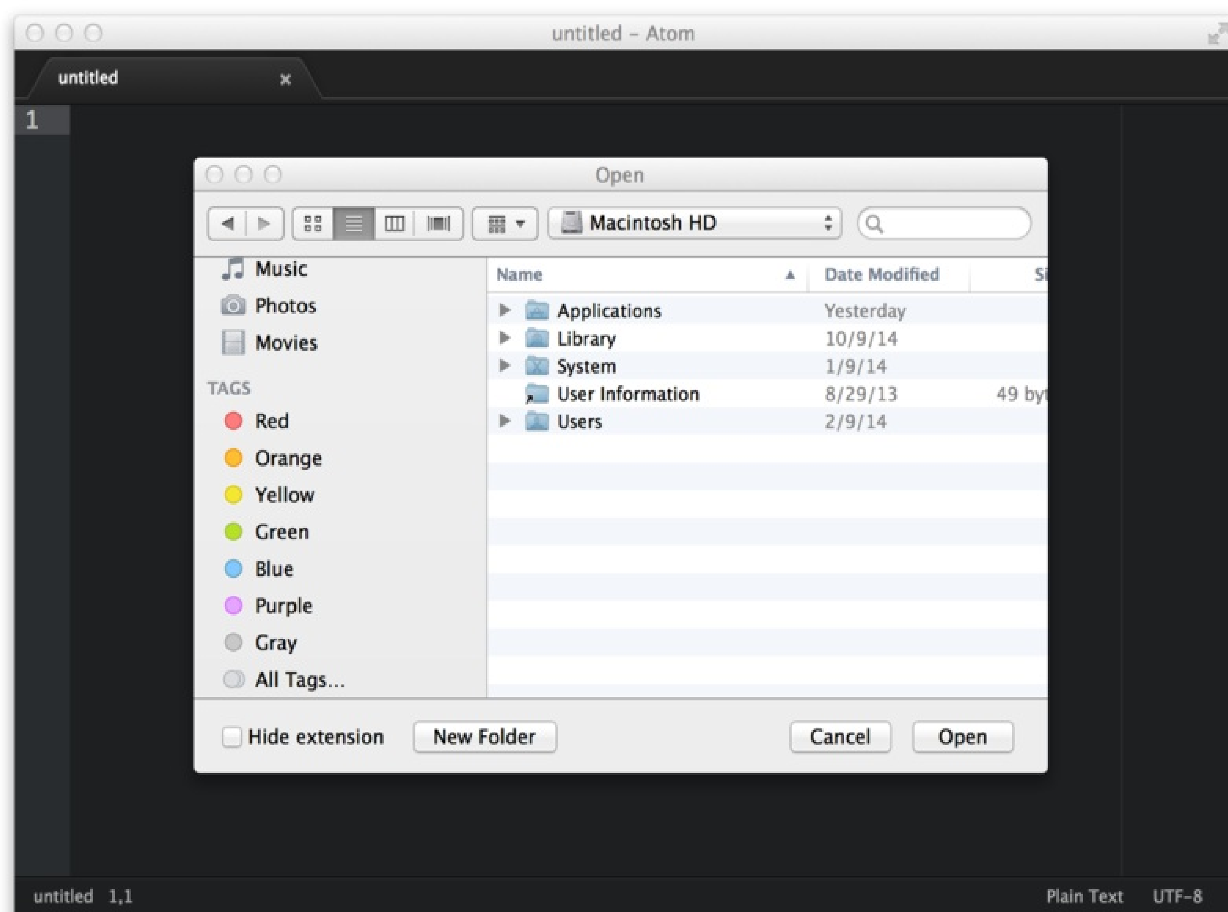
你也可能仅仅是因为期待即将到来的新功能，所以时不时来这里看一看。

打开、编辑、保存文件

现在我们已经设置好了编辑器，让我们来了解一下如何打开和编辑文件吧，毕竟这才是一个文本编辑器真正的功能。

打开文件

在 Atom 中有几种方式可以打开一个文件。你可以在菜单栏中点击 File 菜单下的 Open, 或者用快捷键 `cmd-O`，用操作系统的对话框选择一个文件。



这在打开一个不属于当前项目的文件非常有用，或者更可能是你刚打开一个新的窗口。

另一种打开文件的方法是用命令行。在 Atom 的菜单栏中有一个名为 Install Shell Commands 的菜单项，他会向你的终端中安装一个新的名为 `atom` 的命令，你可以用一个或多个文件路径作为参数去运行 `atom` 命令。

```
$ atom -h
Atom Editor v0.152.0

Usage: atom [options] [path ...]

One or more paths to files or folders may be specified. If there is an
existing Atom window that contains all of the given folders, the paths
will be opened in that window. Otherwise, they will be opened in a new
window.

...
```

这对于从终端中打开一个文件来说非常有用，只需运行 `atom [files]` 即可。

编辑和保存文件

你可以非常简单地编辑一个文件，你只需要用你的鼠标点击一个位置，然后用键盘输入内容即可，Atom 没有特殊的命令或快捷键来进入「编辑模式」。

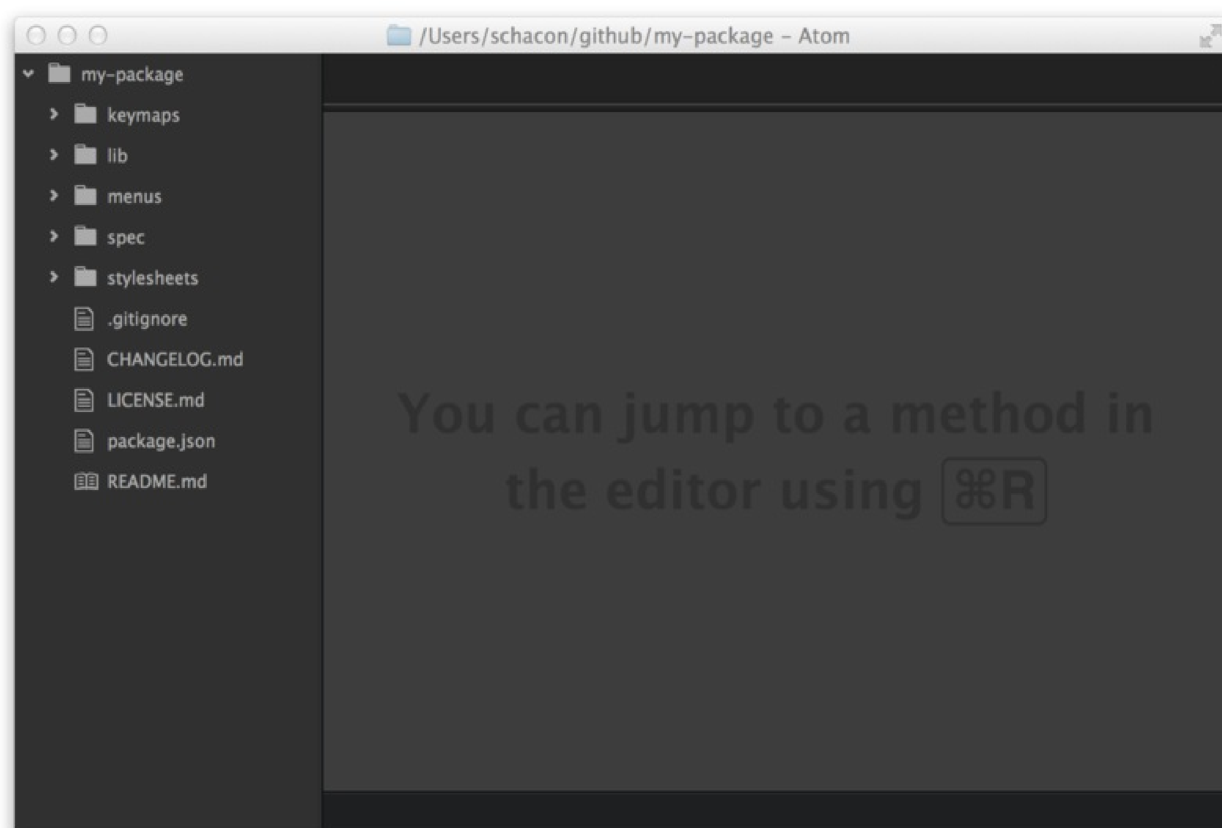
你可以用菜单栏的 File >> Save 或快捷键 `cmd-S` 来保存文件，或者你可以用 Save As 或 `cmd-shift-S` 将文件另存为到另一个路径。最后，你可以用 `ctl-shift-S` 快捷键一次保存 Atom 中所有打开的文件。

打开目录

Atom 不仅仅可以编辑单个文件；大多数情况下你需要编辑由若干个文件组成的项目（Project）。你可以在菜单栏 File >> Open 弹出的对话框中选择一个目录，或者你也可以通过 File >> Add Project Folder... 或快捷键 `cmd-shift-O` 在一个窗口中打开多个目录。

你也可以在命令行下，将多个路径作为参数传递给 `atom`. 例如 `atom ./hopes ./dreams` 会让 Atom 同时打开 hopes 和 dreams 这两个目录。

当你用 Atom 打开一个或多个目录时，目录树会自动地出现在窗口左侧。



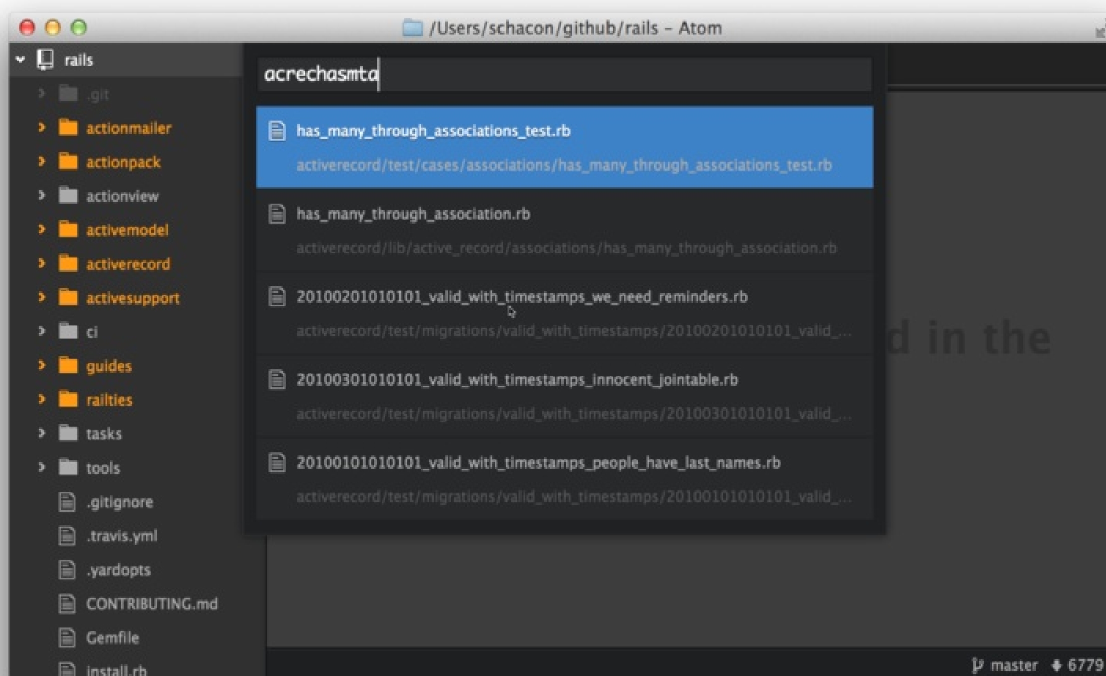
目录树允许你查看和修改当前项目的目录结构，你可以在目录树中打开文件、重命名文件、删除文件、创建文件。

你可以通过快捷键 cmd-\ 或 `tree-view:toggle` 命令来隐藏或重新显示目录树，用快捷键 `ctrl-0` 可以将焦点切换到目录树。当焦点位于目录树上时，你可以用快捷键 `a` 、 `m` 以及 `delete` 来创建、移动或删除文件和目录。你还可以简单地在目录树中用右键点击文件，你可以看到更多选项，除了前面提到的，还可以在操作系统的文件浏览器中显示文件、复制文件的路径到剪贴板。

打开项目中的文件

当你在 Atom 中打开了一个项目（即目录）后，你就可以简单地查找并打开来自项目中文件了。

当你按下 `cmd-T` 或 `cmd-P` 的时候，模糊查找框（Fuzzy Finder）就会弹出。它允许你通过输入文件名或路径的一部分，在整个项目中模糊查找相应的文件。



你也可以通过 `cmd-B` 来只查找已经打开的文件，而不是所有文件。你还可以用 `cmd-shift-B` 来只查找从上次 Git 提交之后修改过或新增的文件。

模糊查找框会根据 `core.ignoredNames` 和 `fuzzy-finder.ignoredNames` 这两个选项来决定不查找哪些文件。如果在你的项目里有很多你不希望它们出现在模糊查找框的文件，那么你可以在选项中添加它们的路径或使用通配符。你可以在设置界面的 Core Settings 下找到这两个选项，之后我们会在 Basic Customization 一节中介绍更多的选项。

这些选项的通配符功能由名为 minimatch 的一个 Node.js 库提供，你可以在这里了解到它的语法：https://github.com/isaacs/minimatch

在 core.excludeVcsIgnoredPaths 这个选项被开启时，模糊查找框会忽略 .gitignore 中指定的的文件，你可以在设置界面中修改这些选项。

模块化的 Atom

就像 Atom 的其他很多部分一样，目录树也并非直接内建在 Atom 中，它是一个独立的插件，被捆绑在 Atom 发行版中并默认启用。

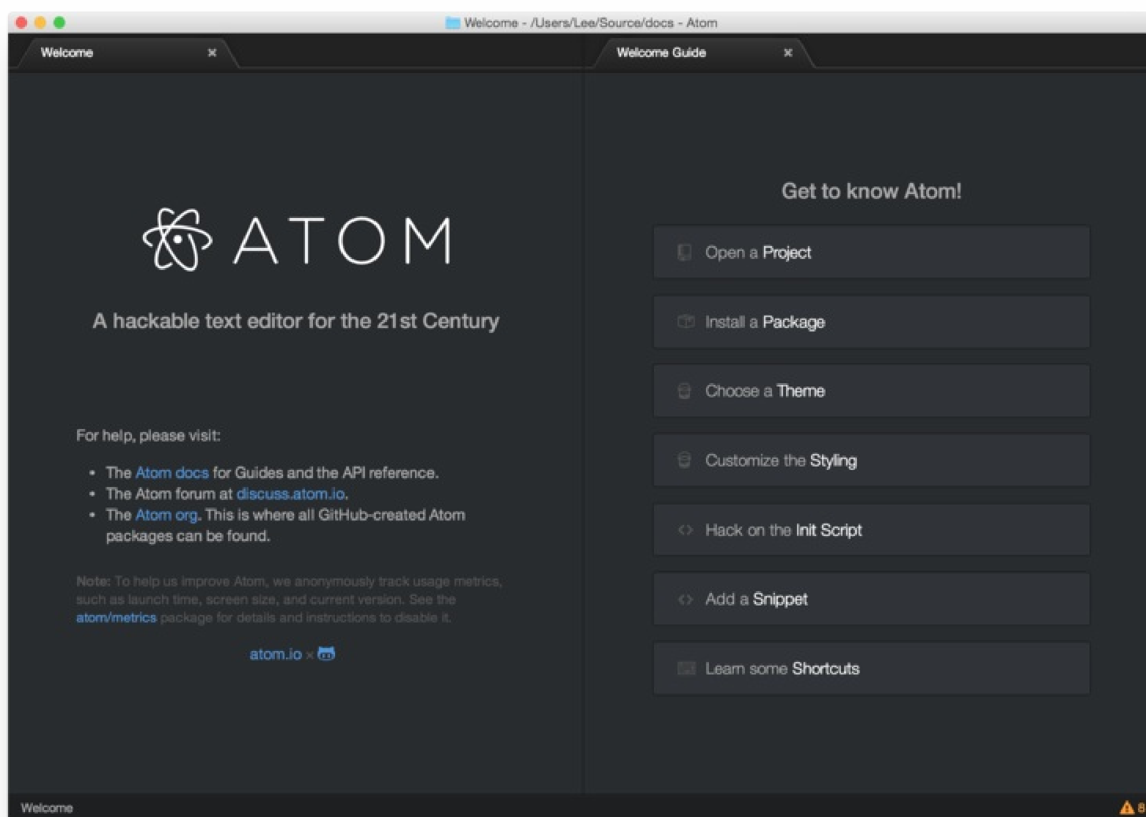
你可以在这里找到目录树插件的源代码：https://github.com/atom/tree-view

Atom 有趣的部分之一就是，很多核心功能实际上只是一个普通的插件——你也可以用类似的方式来实现其他功能。这意味着如果你不喜欢默认的目录树，你完全可以简单地自己编写一个，然后将默认的目录树替换掉。

Atom基础

既然Atom在你的系统中已经安装了，让我们启动、配置并且熟悉这一编辑器吧。

当你首次启动Atom时，你会看到这样一个界面：



这是Atom的欢迎界面，它会给你一些很好的建议，关于如何开始使用这个编辑器。

基本的术语

首先，让我们熟悉一些在这篇文档中使用的基本的术语。

缓冲区

缓冲区是Atom中一个文件的文本内容。在大多数描述中，它基本类似于文件，但是它是Atom在内存中存放的版本。例如你可以修改文本缓冲区的内容，但是如果你不保存文件，它就不会写到相关的文件中。

面板（**pane**）

面板是Atom中可见的部分。如果你去看我们刚才加载的欢迎界面，会看到四个面板 —— 标签栏（tab bar），行号栏（gutter，行号在里面），底部的状态栏（status bar），最后是文本编辑器。

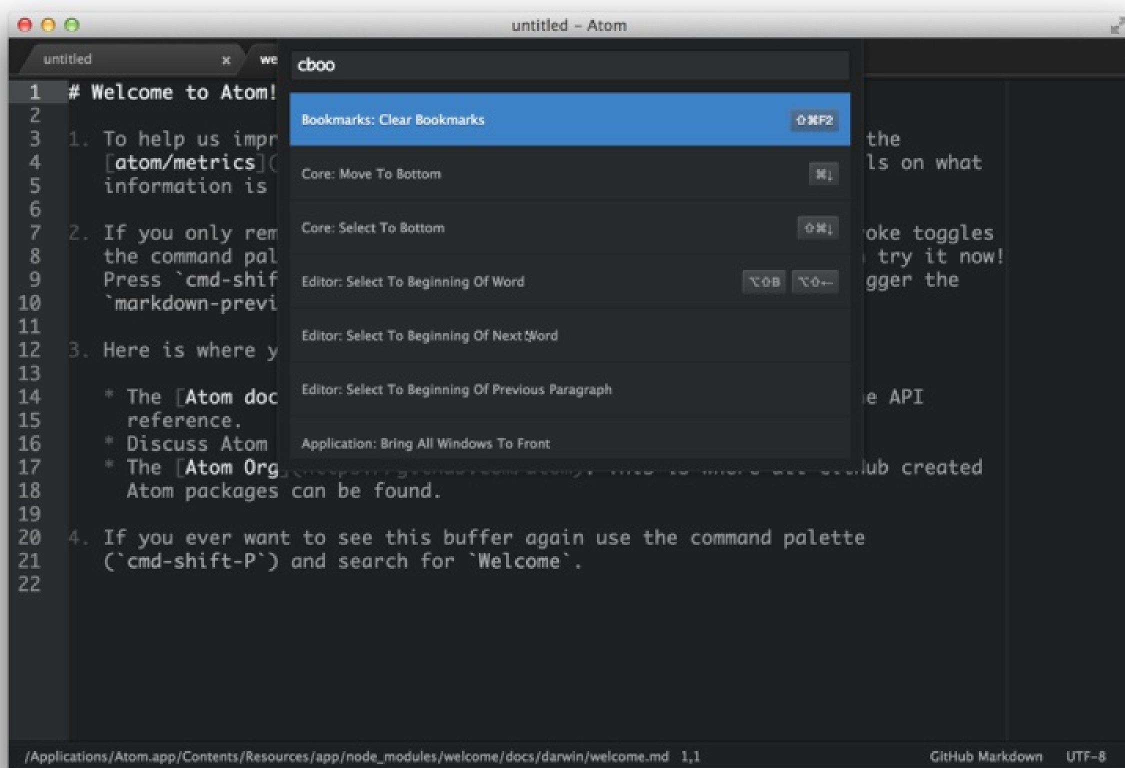
命令面板（Palette）

在欢迎界面中，我们介绍了Atom中最重要的命令，“命令面板”。如果在编辑器获得焦点时，按下 `cmd-shift-P` ，就会弹出命令面板。

在整个教程中我们使用类似 `cmd-shift-P` 的快捷键来演示如何执行命令。这些是Atom在Mac上的默认快捷键。它们有时候会有些差异，取决于你的平台。

你可以使用命令面板来查找正确的快捷键，如果它由于一些原因没有生效。

这一搜索驱动的菜单可以执行Atom中几乎任何主要的工作。你可以按下 `cmd-shift-P` 来搜索命令，而不是在应用的菜单上点来点去来寻找东西。

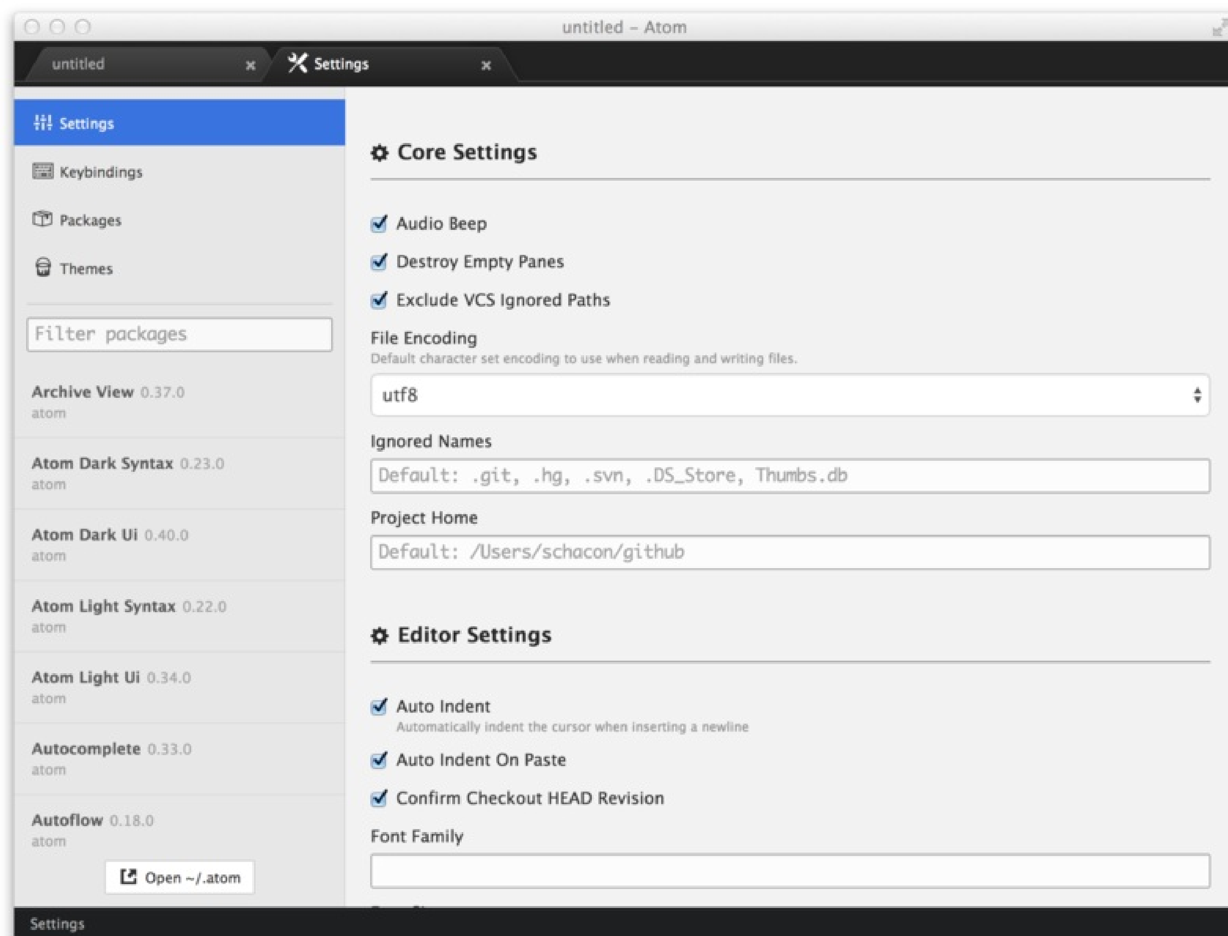


你不仅仅可以在上千种命令中快速查看和查找，也可以查看一个命令上是否有关联的快捷键。这是非常不错的，因为它意味着你能够以自己的方式做一些有趣的事情，并且同时记住使用它的快捷键。

在这篇教程的剩余部分，我们尝试简单讲述一下你可以在命令面板搜索到的各种文本，除了不同命令的快捷键。

设置和偏好

在设置界面中，Atom提供了许多你可以修改的设置和偏好。

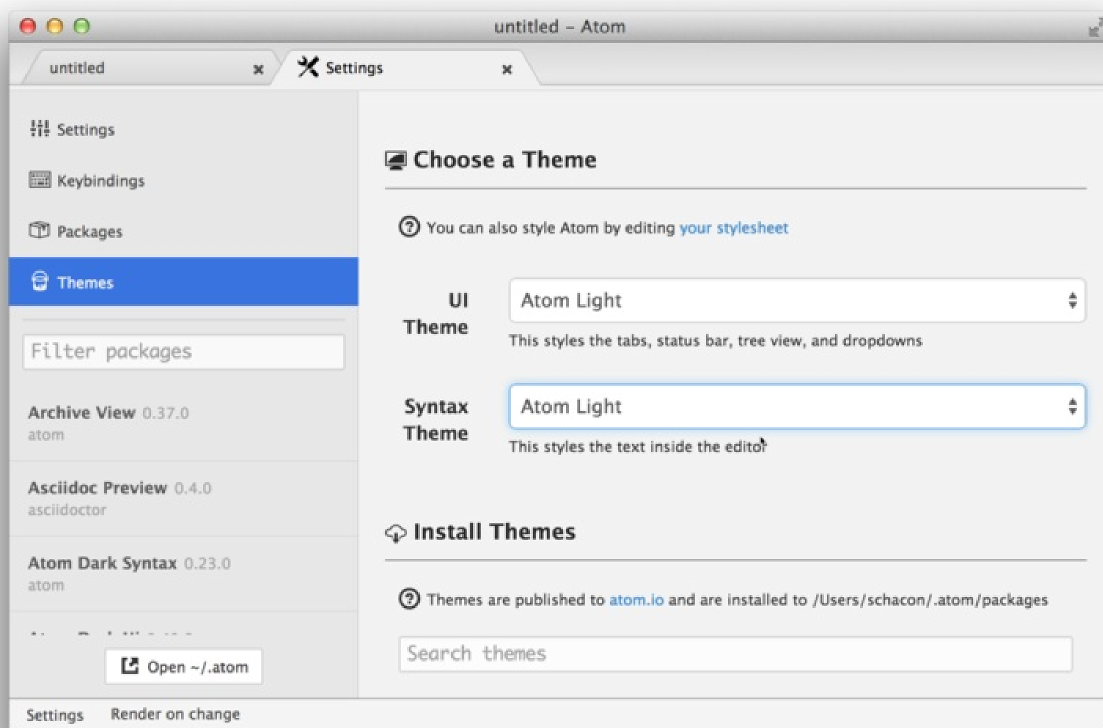


这包括调整配色和主题、指定如何处理换行、字体设置、tab宽度、滚动速度、和一些其它的设置。你也可以使用这个界面来安装新的主题和包，这会在“Atom中的包”一章提到。

你可以通过菜单栏中，“Atom”菜单底下的“Preferences”菜单项，来打开这个设置界面。你也可以在命令面板中搜索 `settings-view:open` 命令，或者按下 `cmd-,` 快捷键。

修改颜色主题

设置视图也允许你为Atom修改颜色主题。Atom自带4种不同的UI颜色主题，亮色或者暗色调的Atom和One主题。同时也拥有八种不同的语法颜色主题。你可以通过点击设置视图边栏上的“Themes”菜单，修改当前的主题，或者安装新的主题。



UI主题会修改UI元素的颜色，例如标签页和树视图，而语法主题会修改你加载进编辑器的文本的语法高亮。简单地在下拉列表中选择一个不同的项来修改主题。

软换行（Soft Wrap）

你可以在设置视图中指定空白字符和软换行的偏好。

- [ ] Show Indent Guide
- [ ] Show Invisibles
- [x] Show Line Numbers
- [x] Soft Tabs
- [x] Soft Wrap
- [ ] Soft Wrap At Preferred Line Length

Tab Length

2

开启“Soft Tabs”（软tab字符）会在你按下 `tab` 键的时候插入空格来替代真正的tab字符。“Tab Length”指定了要插入多少个空格，或者“Soft Tabs”禁用时tab字符用多少个空格来表示。

“Soft Wrap”（软换行）选项会在一行的长度超出编辑器宽度的时候将其换行。如果这一项被禁用，那一行会超出编辑器的边框，你只能通过滚动窗口来查看剩下的内容。如果“Soft Wrap At Preferred Line Length”被选中，一行会在超过80个字符的地方换行，而不是编辑器的宽度。你也可以把一行默认的长度修改成不是80的别的值。

在“基本的自定义”一章中，我们将会看到如何为不同的文件类型指定不同的换行偏好（例如你想在Markdown文件中自动换行，但是代码文件中不这样）。

Beta功能

由于Atom已经开发完成了，所以有时有一些新的功能在发布给每个人之前会被测试。在一些情况中，这些变更默认是关闭的，但是可以在设置视图中打开，如果你想要尝试它们的话。

- [x] Use Hardware Acceleration
  Disabling will improve editor font rendering but reduce scrolling performance.
- [ ] Use Shadow DOM
  Enable to test out themes and packages with the new shadow DOM before it ships by default.
- [ ] Zoom Font When Ctrl Scrolling
  Increase/decrease the editor font size when pressing the Ctrl key and scrolling the mouse up/down.

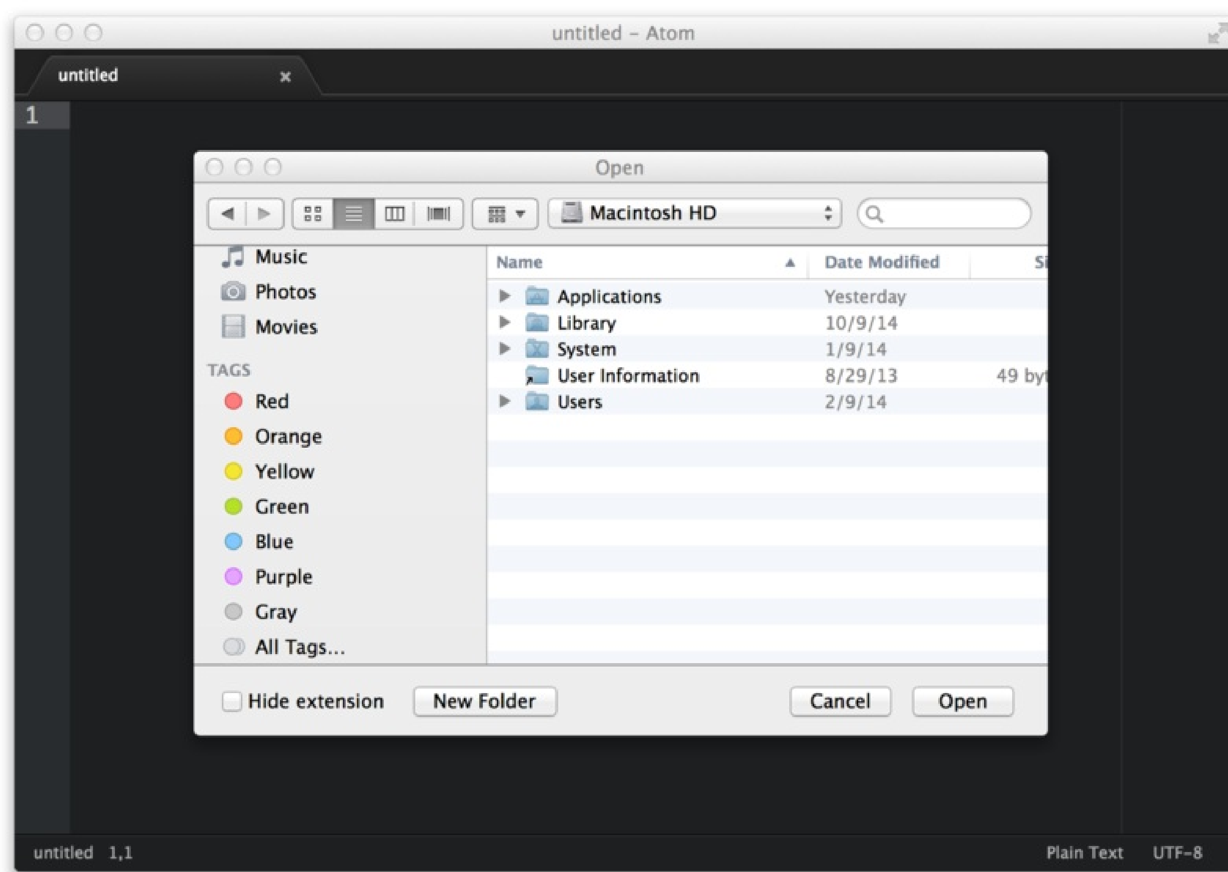
包的开发者为了确认他们的包仍旧在新的功能下生效，会在它们发布给大多数用户之前接触这些特性或者变更，这主要对他们比较有用。然而，如果你对这些新的特性比较感兴趣，偶尔尝试它们中的一些也是十分有趣的。

打开、修改和保存文件

既然你的编辑器看起来并且表现出你想要的样子，让我们来打开并编辑文件。毕竟这是一个文本编辑器，对不对？

打开文件

在Atom中打开文件有许多种方法，你可以在菜单栏选择“File >> Open”，或者按下 `cmd-O` 来从系统对话框中选择一个文件。



这对于打开不在你当前项目（接下来会讲到）中的文件，或者由于一些原因你想打开一个新的窗口，是十分有用的。

在Atom中打开文件的另一种方式，是在命令行中使用 `atom` 命令。如果你使用Mac，Atom的菜单栏有个命令叫做“Install Shell Commands”，它会安装 `atom` 和 `apm` 命令，如果Atom自己不能安装它们的话。在Windows或者Linux上面，这两个命令作为Atom安装进程的一部分自动安装。

你可以使用`atom`带着一个或者多个文件目录来在Atom打开这些文件。

```
$ atom -h
Atom Editor v0.152.0

Usage: atom [options] [path ...]

One or more paths to files or folders may be specified. If there is an
existing Atom window that contains all of the given folders, the paths
will be opened in that window. Otherwise, they will be opened in a new
window.

...
```

如果你熟悉控制台或者使用它完成很多工作，这是相当好用的工具。只是执行`atom [files]`命令，你就可以开始编辑了。

编辑和保存文件

编辑文件很直接，你可以使用鼠标点击和滚动，以及打字来修改内容。Atom中没有特殊的编辑模式或者快捷键。

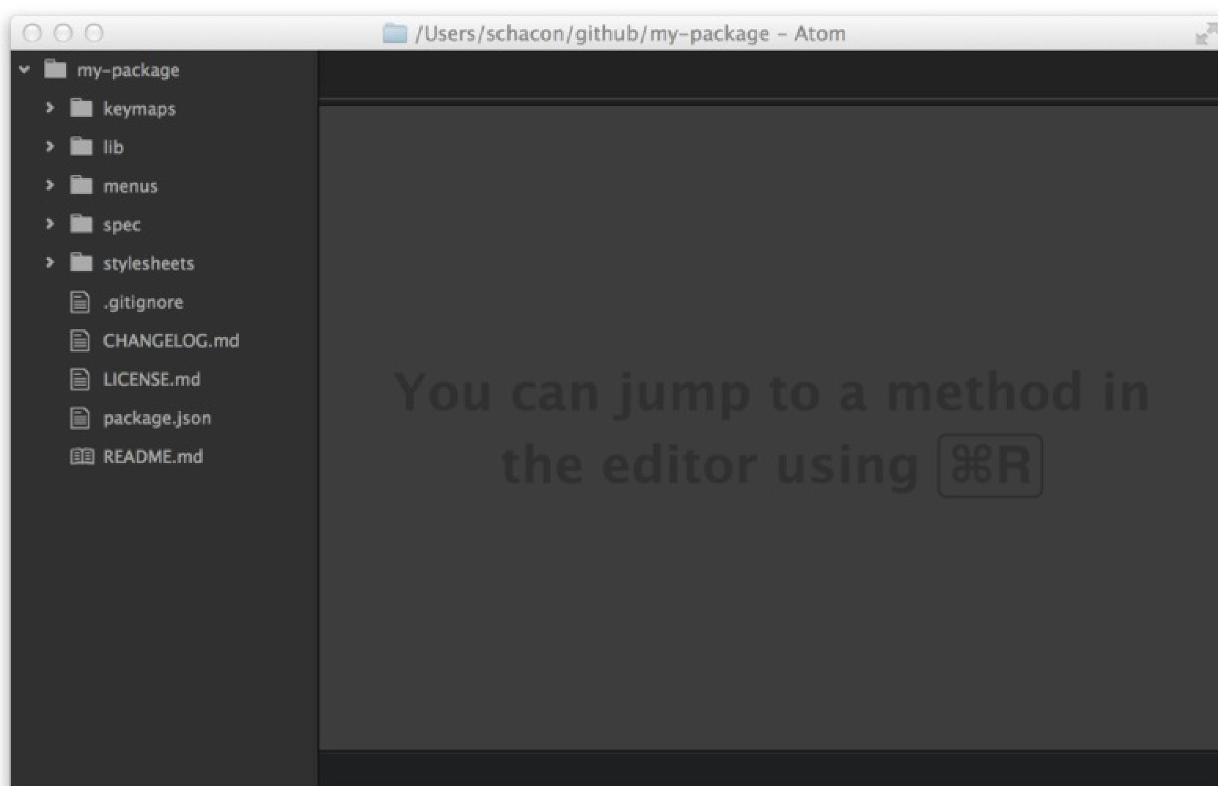
你可以从菜单栏选择“File >> Save”，或者`cmd-s`快捷键来保存文件。如果你选择了“Save As”，或者按下了`cmd-shift-s`快捷键，就可以将当前编辑器中的内容保存到一个不同的文件名下面。最后，你可以按下`cmd-alt-s`来保存你在编辑器中打开的所有文件。

打开目录

Atom并不只能够处理单个文件，你可能大多数时间都花在处理多个文件的项目。从菜单栏选择“File >> Open”，并且从对话框中选择一个目录来打开目录。你也可以从菜单栏选择“File >> Add Project Folder…”，或者按下`cmd-shift-O`快捷键，在你当前的Atom窗口中添加不止一个目录。

你可以在命令行中打开任意数量的目录，通过向`atom`命令传递它们的路径。例如你可以运行`atom ./hopes ./dreams`命令，来同时打开`hopes`和`dreams`目录。

当你在Atom中打开一个或者多个目录时，Atom会自动在窗口的一边显示树视图。



树视图允许你浏览和修改文件以及你项目的目录结构。你可以从这个视图中打开、重命名、删除和创建新的文件。

你也可以使用 `cmd-\` 或者命令面板的 `tree-view:toggle` 命令来隐藏和显示它。以及 `ctrl-0` 来在它上面设置焦点。当树视图具有焦点时，你可以按下 `a` 、 `m` 、 `d` 来添加、修改和删除文件和文件夹。你可以在树视图中简单地右键点击文件和文件夹，来查看许多不同的选项，包括在你的本地文件系统中展示文件，或者复制文件路径到你的剪贴板。

Atom 模块

像许多Atom的部分一样，树视图并不直接构建在编辑器内，而是Atom默认自带的独立的包中。

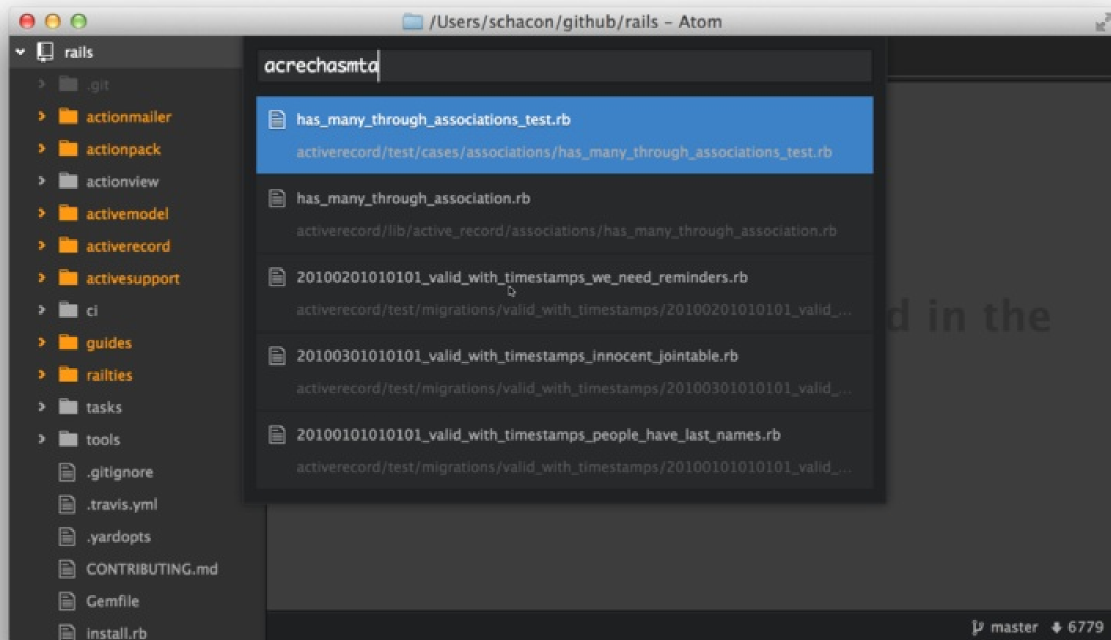
你可以在这里找到树视图的代码。

这是Atom的有趣的事情之一。许多核心功能实际上只是包，它们和你实现一些其它功能的方式相同。这意味着例如你不喜欢树视图，你可以非常简单地编写你自己对于该功能的实现，并且完全替换它。

在项目中打开文件

你在Atom中打开一个项目之后，你可以轻松地找到并且打开项目中的文件。

如果你按下 `cmd-T` 或者 `cmd-P` ，模糊查找工具的对话框就会弹出。这样能够让你通过输入路径的一部分，在项目中的任何目录中寻找任何文件。



你也可以使用 `cmd-B` 只在当前打开的文件中搜索（而不是项目中的每个文件）。这样的搜索只在“缓冲区”或者打开的文件中进行。你也可以使用 `cmd-shift-B` 来限制模糊查找工具，只在上次Git提交以来添加和修改的文件中搜索。

模糊查找工具同时使用 `core.ignoredNames` 和 `fuzzy-finder.ignoredNames` 设置，来过滤不显示的文件和文件夹。如果你的项目中有大量你不想在其中搜索的文件，你可以向这两个设置之一添加通配符或者目录。我们将会在“全局设置”中了解设置的更多细节，但是现在你可以在设置视图的核心设置下面简单地设置它。

这两个设置会解释为Unix通配符，在minimatch Node.js库中实现。

你可以在这里阅读更多关于minimatch的细节。

当 `core.excludeVcsIgnoredPaths` 开启的时候，并不会展示Git忽略的文件。你可以在设置视图中切换它，它是顶级选项之一。

小结

你应该对Atom是什么，以及你想使用Atom做什么有了基础的了解。你也应该把它保留在你的系统中，并使用它完成更多基本的文本编辑操作。

现在，请准备好深入探索这一有趣的工具。

使用Atom

在我们介绍完Atom最基础的部分之后，我们要了解如何真正尽可能使用它了。在这一章中我们会介绍如何为了添加新功能而寻找并安全新的包，如何寻找并安装新的主题，如何以一种更高级的方法处理文本，如何以任何你想要的方式自定义编辑器，如何使用git做版本控制，以及其它。

Atom中的包

首先，让我们从Atom的包系统开始讲起。像我们前面提到过的那样，Atom自己只是一个非常基础的功能核心，它上面加载了许多有用的包，这些包添加新的功能，像树视图（Tree View）和设置视图（Settings View）。

实际上，默认情况中，Atom中所有的功能由超过70种包组成。例如，你在首次启动Atom时看到的欢迎对话框，拼写检查工具，主题和模糊查找工具都是独立的包，它们使用了你所访问的相同API。我们在第三章将会看到更多细节。

这意味着所有包都可以变得越来越强大，并且它们可以改变任何东西，从整体接口的外观和感觉，到核心功能的基本操作。

要想安装一个新的包，你可以使用设置视图中的install选项卡，现在你已经非常熟悉了。简单地打开设置视图（ `cmd-,` ），点击“install”选项卡，并且在“Install Packages”下面输入你要查找的东西，那个地方提示“Search Packages”。

列在底下的是发布到atom.io的包，它是Atom包的官方注册处（registry）。设置面板中的搜索操作，会进入atom.io中的包注册处寻找，之后拉回任何匹配你搜索的东西。



所有的包都会在点击“install”按钮后安装。点击之后会下载并安装相应的包，你的编辑器会拥有那个包提供的功能。

包的设置

在Atom安装了一个包之后，那个包会出现在“Package”选项卡下面的侧面板中，同时带着Atom预先安装的所有包。你可以在"Filter packages by name"文本框中输入内容，来过滤这个列表并找到你想要找的包。

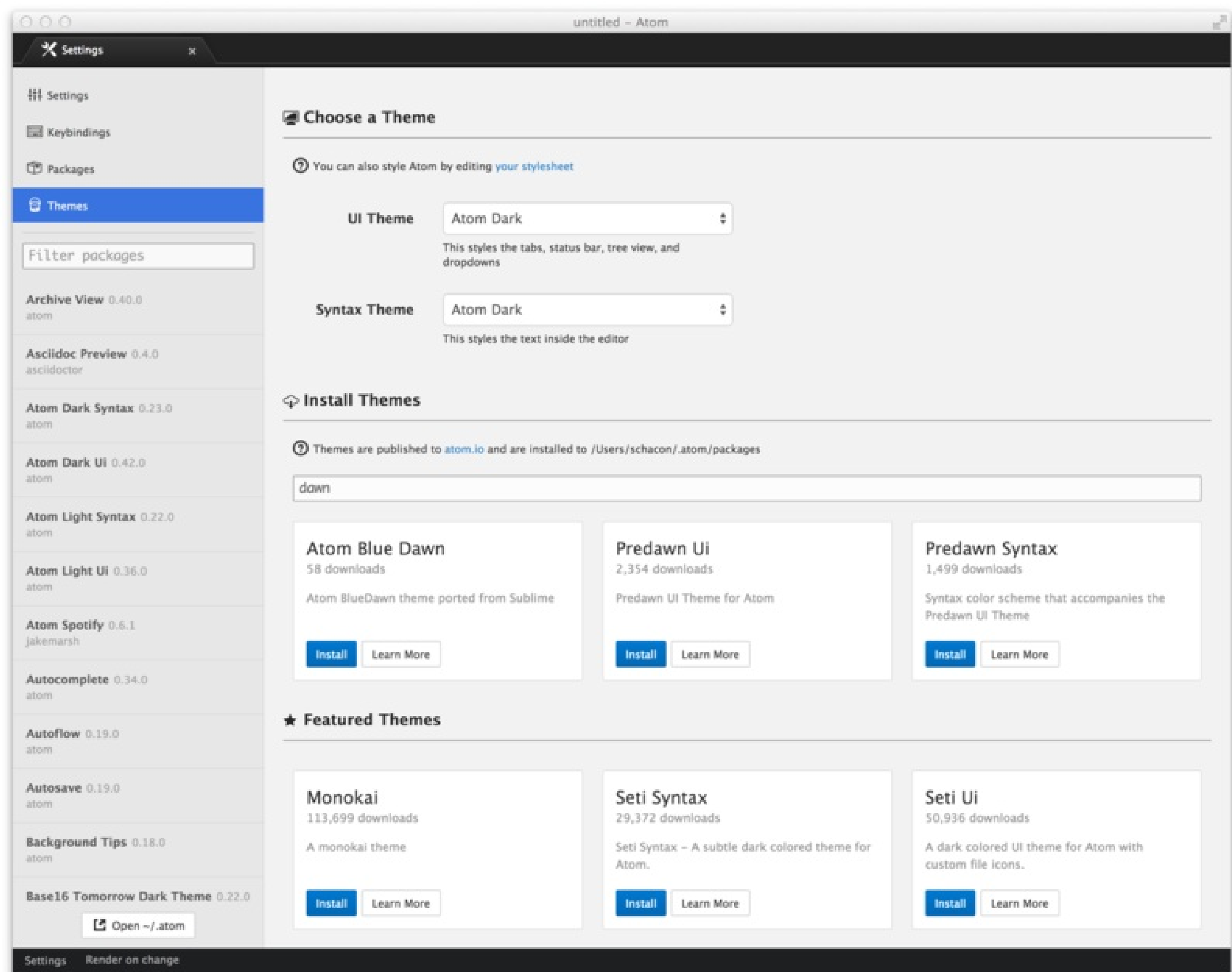


点击一个包的“Settings”按钮会弹出这个包特定的设置窗口。你可以查看它所有的快捷键，暂时禁用这个包，查看它的源码，查看当前版本，报告问题以及卸载这个包。

如果你安装的任何包有新的版本发布，Atom会自动检测它。你可以从当前窗口，或者“Update”选项卡来升级这个包。这有助于你对所有安装的包保持更新。

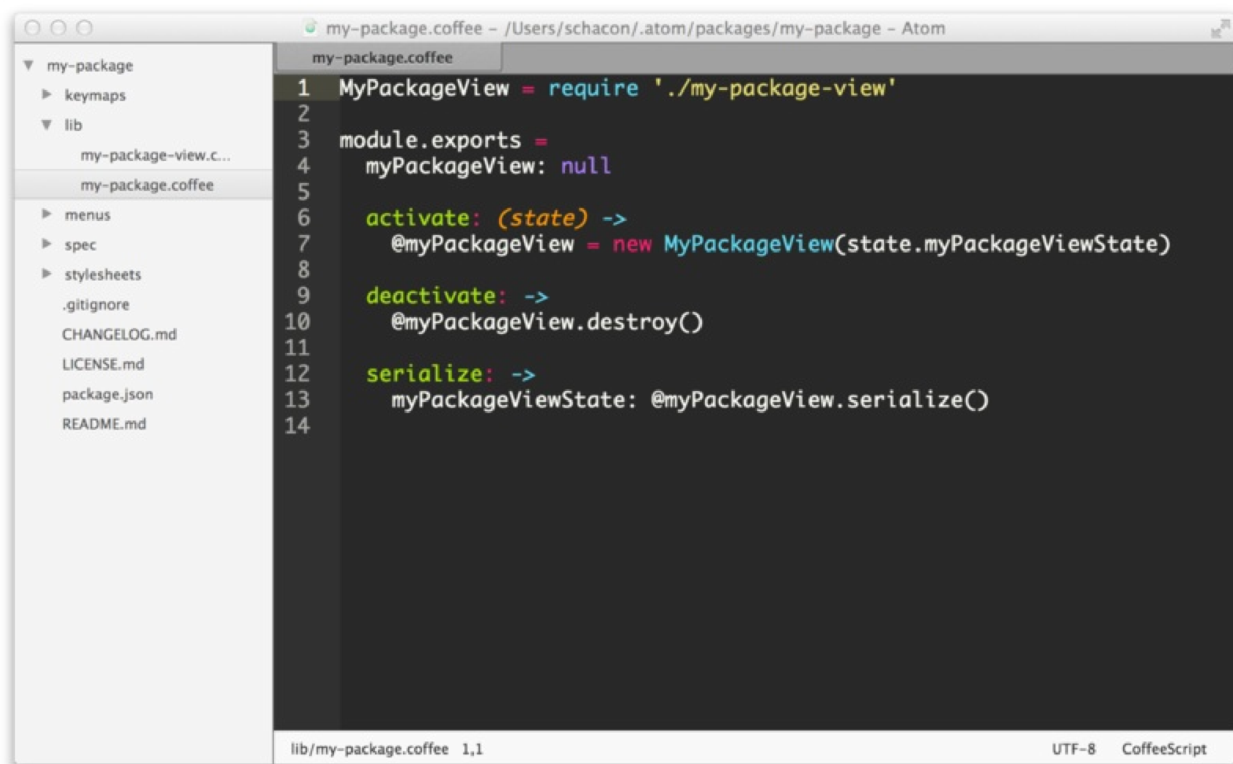
Atom的主题

你也可以从设置视图中，为Atom寻找并安装新的主题。这些主题可以是UI主题，或者语法高亮主题。你可以在“install”选项卡中寻找他们，就像寻找新的包那样。要确保你点击了搜索框旁边的“Themes”切换按钮。



点击主题的标题会弹出它在atom.io上的简介页面，通常会显示它的快照。你可以在安装前看看它是什么样子。

点击“install”按钮会安装该主题，并且在“Theme”下拉框中可供使用。就像我们在“更改主题颜色”一节看到的那样。



命令行

你也可以在命令行中通过 `npm` 安装主题包。

通过在控制台运行一下命令，检查你是否安装了 `apm` ：

```
$ apm help install
```

你会看到一条有关 `apm install` 命令的详细信息打印出来。

如果没有的话，打开Atom，运行 `Atom > Install Shell Commands` 菜单 `apm` 和 `atom` 命令。

你也可以使用 `apm install` 命令安装包：

- `apm install <package_name>` 会安装最新版本。
- `apm install <package_name>@<package_version>` 会安装指定版本。

比如， `apm install emmet@0.1.5` 会安装Emmet包的0.1.5发行版。

你也可以使用 `apm` 寻找新的包来安装。如果你运行 `apm search` 命令，你可以在包注册处搜索想要找的东西。

```
$ apm search coffee
Search Results For 'coffee' (5)
├── coffee-trace Add smart trace statements to coffee files with one keypress each. (77 d
├── coffee-navigator Code navigation panel for Coffee Script (557 downloads, 8 stars)
├── atom-compile-coffee This Atom.io Package compiles .coffee Files on save to .js files.
├── coffee-lint CoffeeScript linter (3336 downloads, 18 stars)
└── git-grep `git grep` in atom editor (1224 downloads, 9 stars)
```

你也可以使用`apm view`查看指定包的详细信息。

```
$ apm view git-grep
git-grep
├── 0.7.0
├── git://github.com/mizchi/atom-git-grep
├── `git grep` in atom editor
├── 1224 downloads
└── 9 stars

Run `apm install git-grep` to install this package.
```

在Atom中移动

用鼠标和方向键，简单地在Atom中移来移去非常容易，然而Atom有一些快捷键，可以让你把手一直放到键盘上，更快速地浏览文档。

首先，Atom自带许多Emacs的快捷键来浏览文档。要想上移或者下移一个字符，你可以按 `ctrl-P` 和 `ctrl-N` 。左移或右移一个字符，按 `ctrl-B` 和 `ctrl-F` 。这样等同于按下方向键，但是一些人不喜欢把他们的手移到方向键的位置。

除了单个字符的移动，还有一些其他的用于移动的快捷键。

`alt-B` , `alt-left`

移动到单词开头。

`alt-F` , `alt-right`

移动到单词末尾。

`cmd-right` , `ctrl-E`

移动到整行末尾

`cmd-left` , `ctrl-A`

移动到整行开头

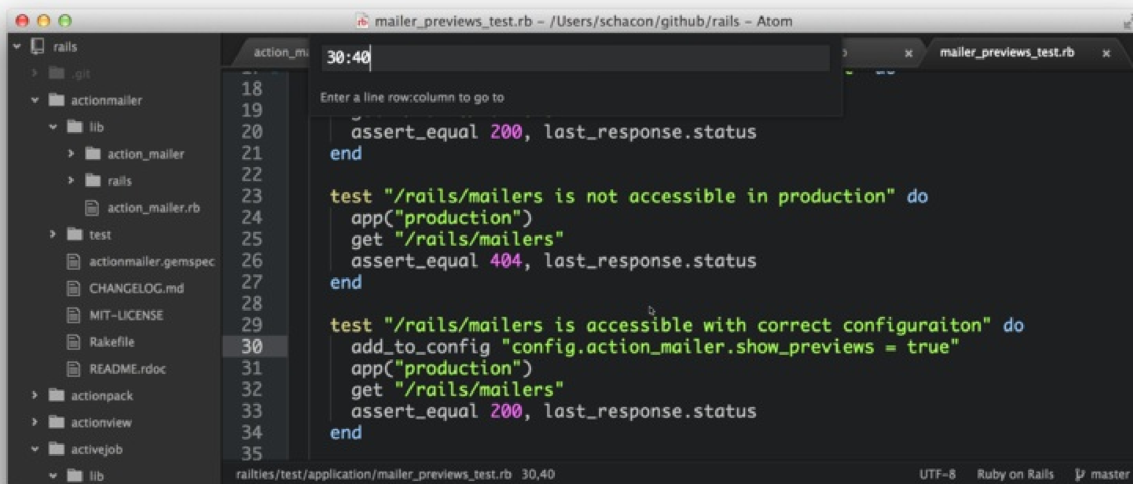
`cmd-up`

移动到文件开头。

`cmd-down`

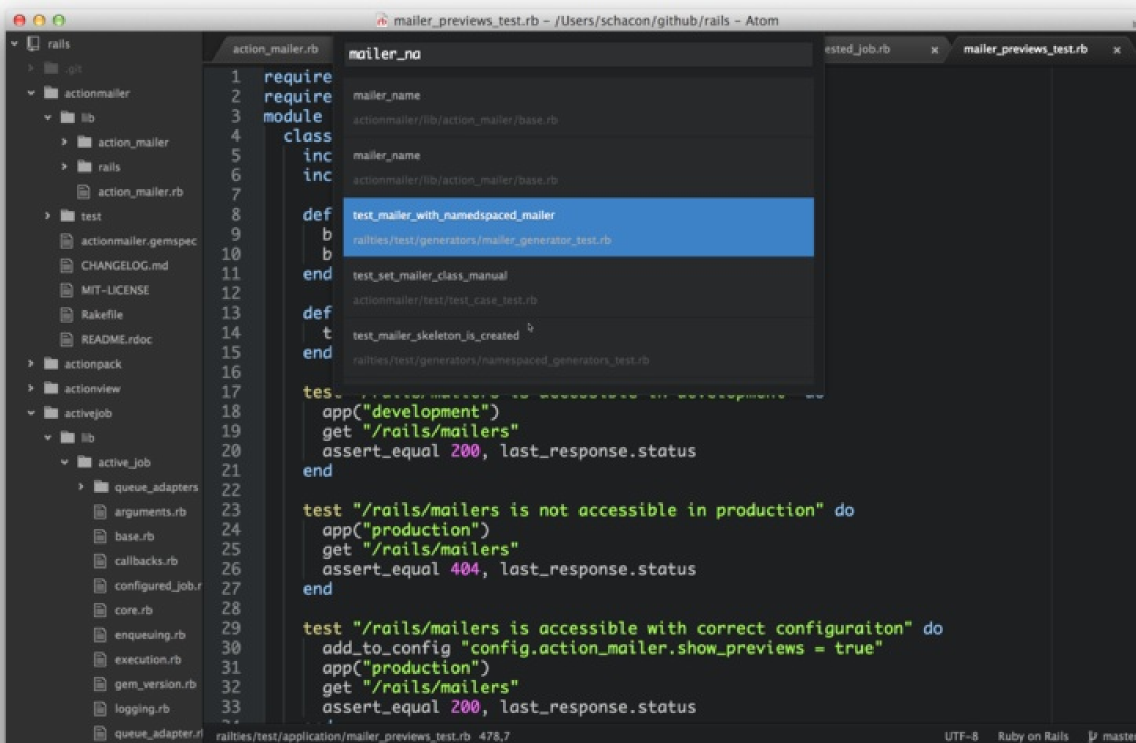
移动到文件末尾。

你也可以使用 `ctrl-G` ，通过指定一行（和列）来直接移动光标。这会弹出一个对话框问你想要跳到哪一行。你同样可以使用 `row:column` 语法来跳到那一行的某个字符。



通过符号浏览

你也可以提供更多的信息来跳转。要想跳到一个方法声明之类的符号，按下 `cmd-r` 。它会打开一个列表，包含当前文件中所有的符号，你可以通过 `cmd-t` 进行模糊查找。使用 `cmd-shift-r` 来查找存在于整个项目中的符号。



你也可以使用 `ctrl-alt-down` 来跳到光标下的方法或者函数声明。

首先，你需要确保你的项目中生成了`tags`（或者`TAGS`）文件。通过安装ctags，并且从命令行中，在你的项目根目录下运行`ctags -R src/`这样的命令，来生成文件。

如果你在Mac中使用Homebrew，运行`brew install ctags`来安装。

你可以通过在你的主目录下生成`.ctags`文件（`~/.ctags`），来自定义tags如何生成。这里是一个例子。

符号浏览功能在atom/symbols-view包中实现。

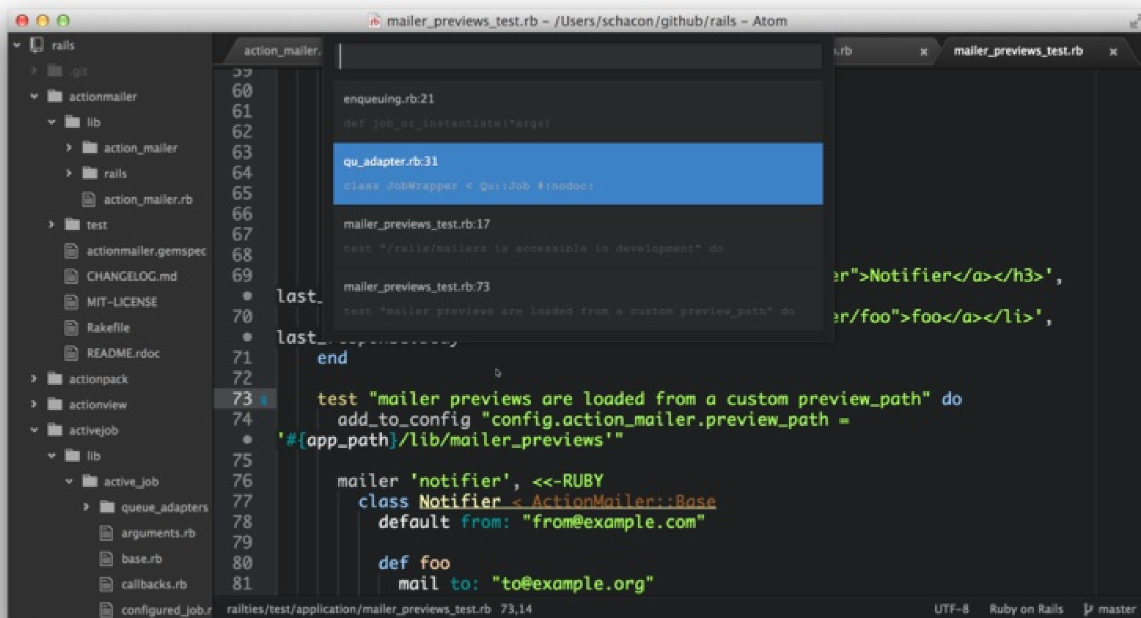
Atom书签

Atom同时拥有一个非常棒的途径，在特定的一行上面加上书签，使你可以快速跳到那一行。

如果你按下`cmd-F2`，Atom会给那一行加上书签。你可以在整个项目中设置一些书签，并且使用它们快速跳到项目中一些重要的行。一个小的书签标识会加在行号后面，像下面这张图的第22行。

按下`F2`之后，Atom会跳到当前文件的下一个书签的位置。如果你按下`shift-F2`则会跳到上一个。

通过按下`ctrl-F2`，你可以看到当前项目中的所有书签的列表，并可以快速筛选它们，跳到任何一个书签的位置。



书签功能在atom/bookmarks包中实现。

文本选择

Atom中的文本选择支持很多操作，比如区域选择、缩进和一些查找操作, 以及用引号或者括号把文字括起来之类的标记操作。

选择命令借鉴了很多查找命令。他们实际上具有相同的快捷键，只不过加了个`shift`。

`ctrl-shift-P`

选择当前位置到上一行的相同列

`ctrl-shift-N`

选择当前位置到下一行的相同列

`ctrl-shift-B`

选择前一个字符

`ctrl-shift-F`

选择后一个字符

`alt-shift-B`, `alt-shift-left`

选择当前位置到单词开头

`alt-shift-F`, `alt-shift-right`

选择当前位置到单词末尾

`ctrl-shift-E`, `cmd-shift-right`

选择当前位置到整行末尾

`ctrl-shift-A`, `cmd-shift-left`

选择当前位置到整行开头

`cmd-shift-up`

选择当前位置到文件开头

`cmd-shift-down`

选择当前位置到文件末尾

除了和移动相关的选择命令，还有一些命令可以选择内容的特定区域。

`cmd-A`

选择整个缓冲区

`cmd-L`

选择整行

`ctrl-shift-W`

选择当前单词

编辑和删除文本

到目前为止，我们介绍了一些用于在文件中移动和选择区域的方法，现在让我们真正来修改一些文本吧。很显然你可以通过打字的方式来输入字符，但是有另一些方法使删除和处理字符变得更简捷易用。

基本操作

有一些用于基本操作的很酷的快捷键，他们十分易用。这些操作包括整行移动文本，整行复制，以及改变大小写。

`ctrl-T`

交换光标两边字符的位置

`cmd-J`

将下一行拼接到当前行的末尾

`ctrl-cmd-up` , `ctrl-cmd-down`

上移或者下移当前行

`cmd-shift-D`

复制当前行

`cmd-K` , `cmd-U`

将当前字符转为大写

`cmd-K` , `cmd-L`

将当前字符转为小写

Atom也带有一个功能，可以对段落重新排版，在超出提供的最大长度的地方硬换行（hard-wrap）。你可以对当前选中区域格式化，使用`cmd-alt-Q`，使其一行的长度不超过80个字符（或者`editor.preferredLineLength`设置为什么都可以）。如果没有选中任何东西，当前段落会被重排。

删除和剪切文本

你也可以从你的缓冲区中剪切或删除文本。不要手下留情。

ctrl-shift-K

删除当前一行

cmd-delete

删除当前位置到整行末尾的内容（在mac中为 `cmd-fn-backspace` ）

ctrl-K

剪切当前位置到整行末尾的内容

cmd-backspace

删除当前位置到整行开头的内容

alt-backspace, alt-H

删除当前位置到单词开头的内容

alt-delete, alt-D

删除当前位置到单词末尾的内容

多光标选择

Atom可以做的一件非常酷的事情，就是支持多个光标，开箱即用。这在处理一个很长的文本列表时会相当有用。

`cmd-click`

添加新的光标

`cmd-shift-L`

将一个多重选择变为多个光标

`ctrl-shift-up` , `ctrl-shift-down`

在当前光标之上或之下添加新的光标

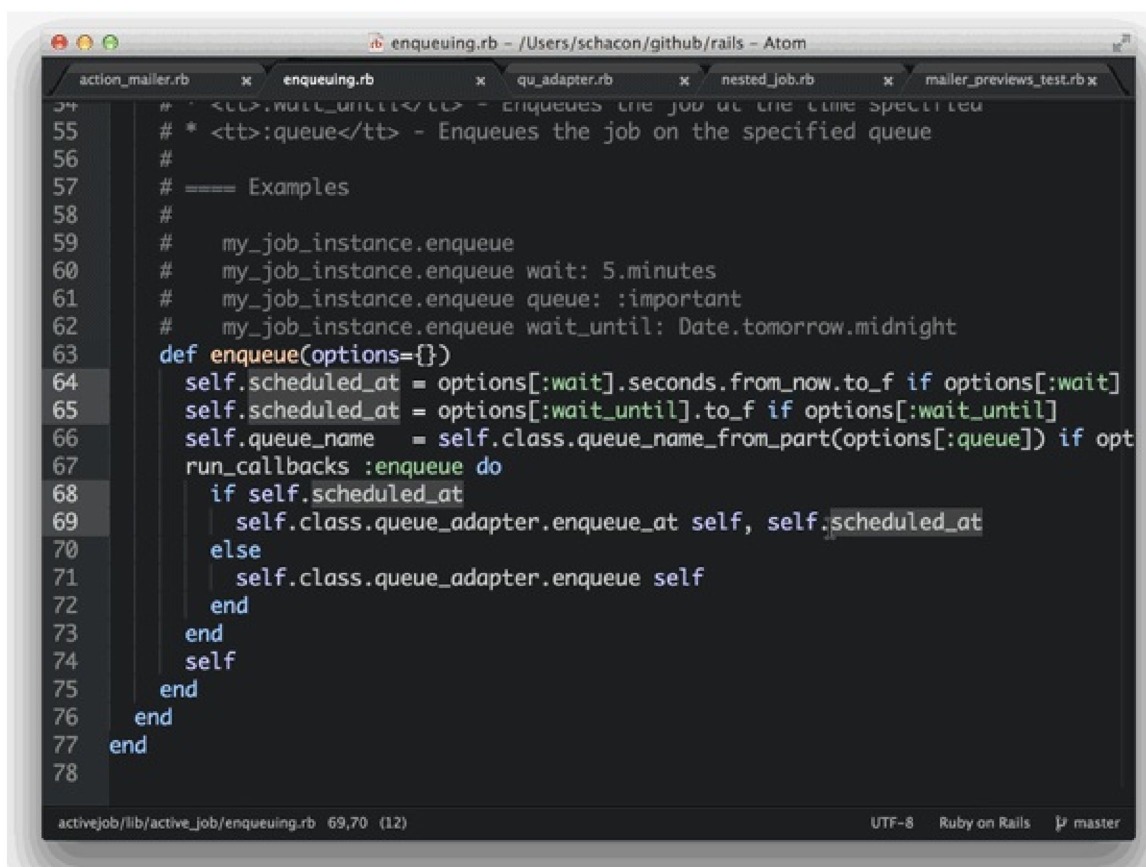
`cmd-D`

选择文档中与当前所选的单词相同的下一个单词

`ctrl-cmd-G`

选择文档中与当前所选的单词相同的所有单词

通过这些命令，你可以在文档的多个位置放置光标，并且一次性有效地在多个位置执行相同操作。



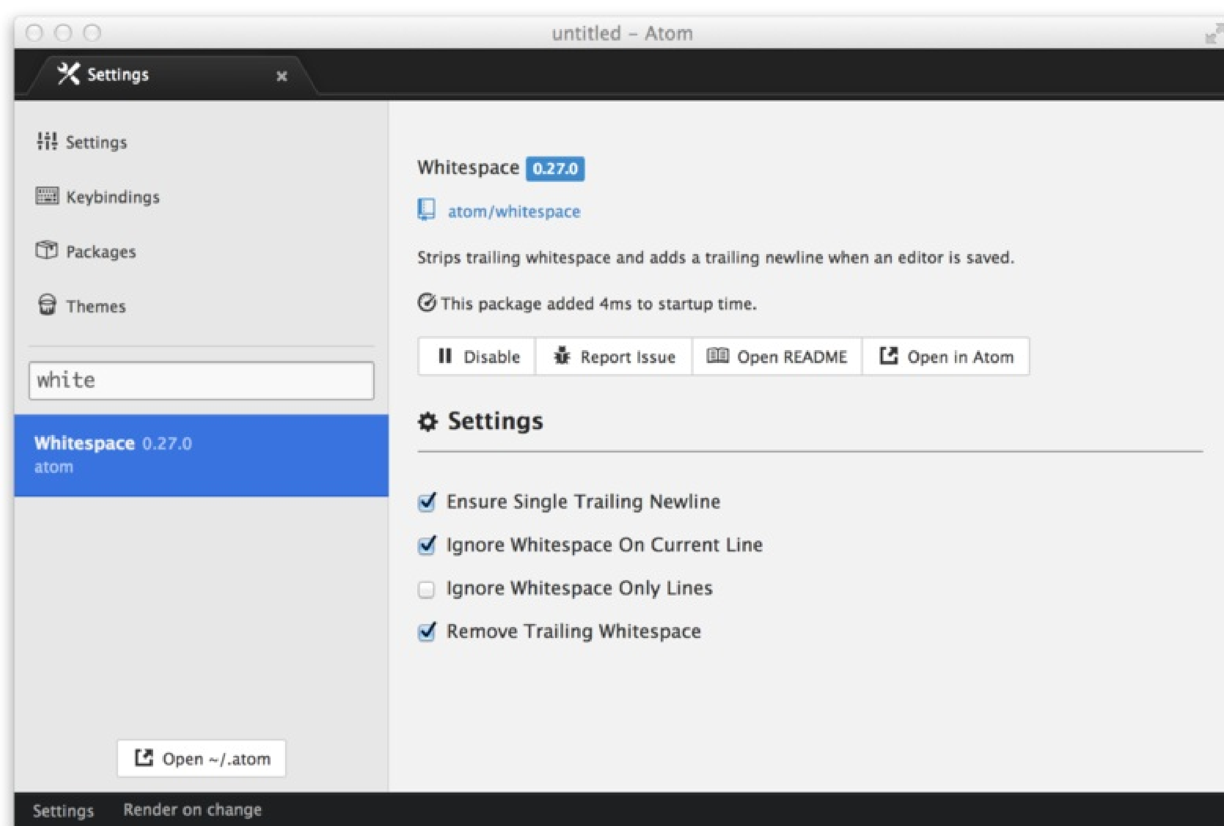
在你执行许多重复性操作时，比如重命名变量或者修改一些文本的格式时，会十分有帮助。你可以一起使用几乎任何插件或命令 —— 比如，修改大小写，以及整行移动或者复制。

你也可以在按下 `command` 键的同时使用鼠标选择文本，来一次性选中多个区域。

空白字符

Atom自带一些命令来帮助你管理你文档中的空白字符。一对非常有用的命令是把空格转换为tab，以及把tab转换为空格。如果你的文档中混用了多种空白字符，这些命令对文档的标准化有巨大帮助。空白字符命令没有相关的快捷键，所以你只能在命令面板中寻找“Convert Spaces to Tabs”（或者反过来），并且选取一个来执行。

空白字符命令在atom/whitespace中实现。空白字符命令的设置，在 `whitespace` 包的页面中提供。



要注意“Remove Trailing Whitespace”选项是默认开启的。意思是每次你保存在Atom中打开的文件时，Atom都会把末尾的空白字符去掉。如果你希望禁用它，在你的设置面板中打开 `whitespace` 包的页面，取消该选项的勾选。

Atom同时也默认确保你文件中有个在末尾的空行，你也可以在上述位置禁用它。

括号

Atom自带一种对括号的智能处理方式。

当你的光标覆盖他们时，Atom会自动高亮 `{}` 、 `()` 和 `[]` 。匹配的xml和html标签也会高亮显示。

Atom也会自动补全 `[]` , `()` , `{}` , `“”` , `''` , `“”` , `‘’` , `«»` , `‹›` 和反引号。当你输入开头的一个时，Atom会补全另一个。如果你在一段选择区域上面输入这些括号或引号的开头，Atom会用对应符号的结尾使区域闭合。

下面是一些其他的有趣的括号相关命令，你可以使用它们。

`ctrl-m`

跳到光标下的括号所匹配的括号。如果没有，就跳到最近的后括号。

`ctrl-cmd-m`

选择当前括号中所有文本

`alt-cmd-.`

闭合当前的xml或html标签。

括号功能在atom/bracket-matcher包中实现。和所有这些包一样，想要修改括号处理相关的默认行为，或者直接禁用它，你可以浏览设置视图（Settings view）中这个包的页面。

编码

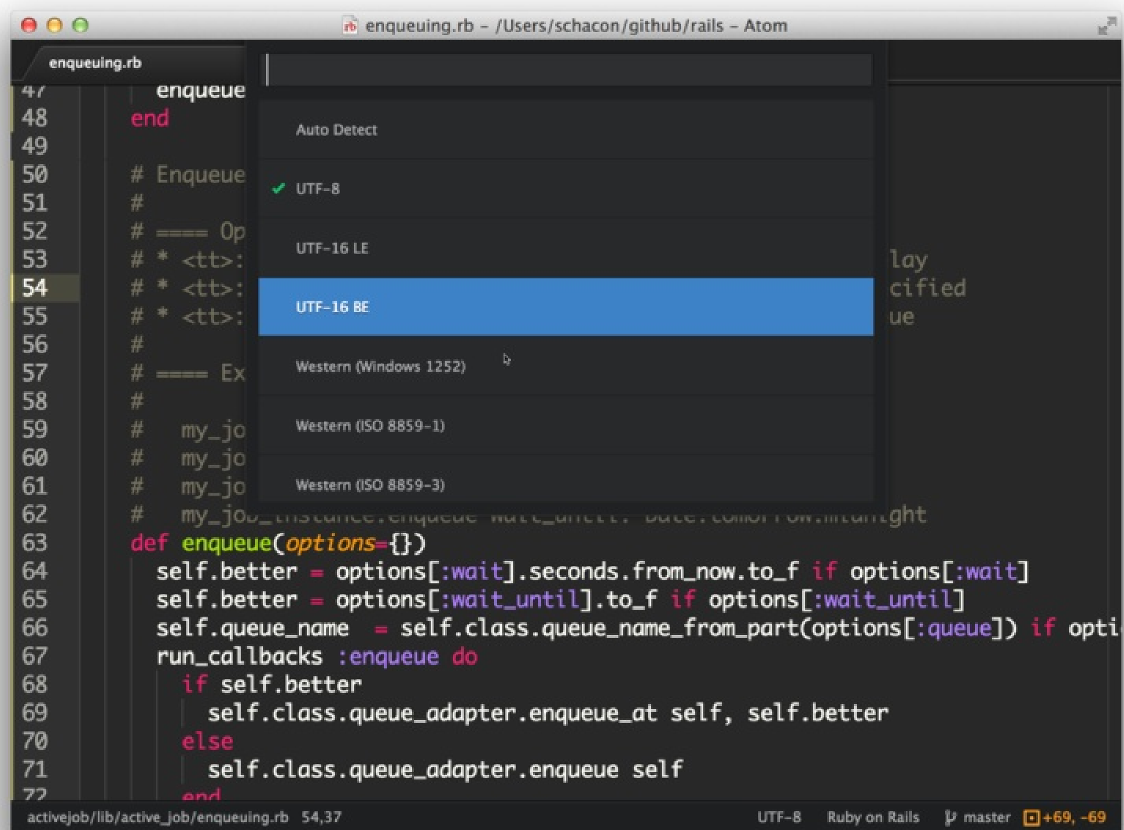
Atom也自带了一些基本的文件编码支持，如果你发现你在处理非UTF-8文件，或者你打算创建这样的文件的话。

`ctrl-shift-U`

拉下菜单来修改文件编码

如果你弹出了编码对话框，你可以选择用来保存文件的编码。

当你打开一个文件时，Atom会自动检测文件编码。如果检测失败，编码会默认设置为UTF-8，它也是新建立的文件的编码。



如果你弹出编码菜单，并且修改了活动编码，文件会在下次保存时以那个编码保存。

编码选择器在atom/encoding-selector包中实现。

查找和替换

在Atom中，对你文件或者项目中的文本进行查找或者替换，非常快速而且容易。

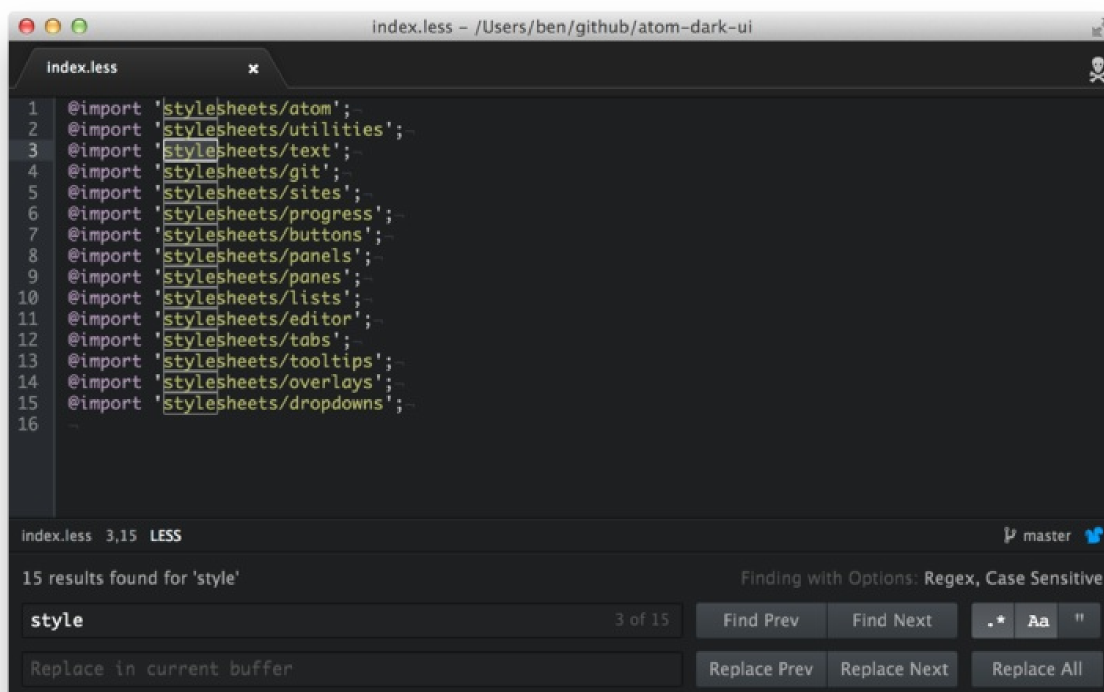
`cmd-F`

在缓冲区中查找

`cmd-shift-f`

在整个项目中查找

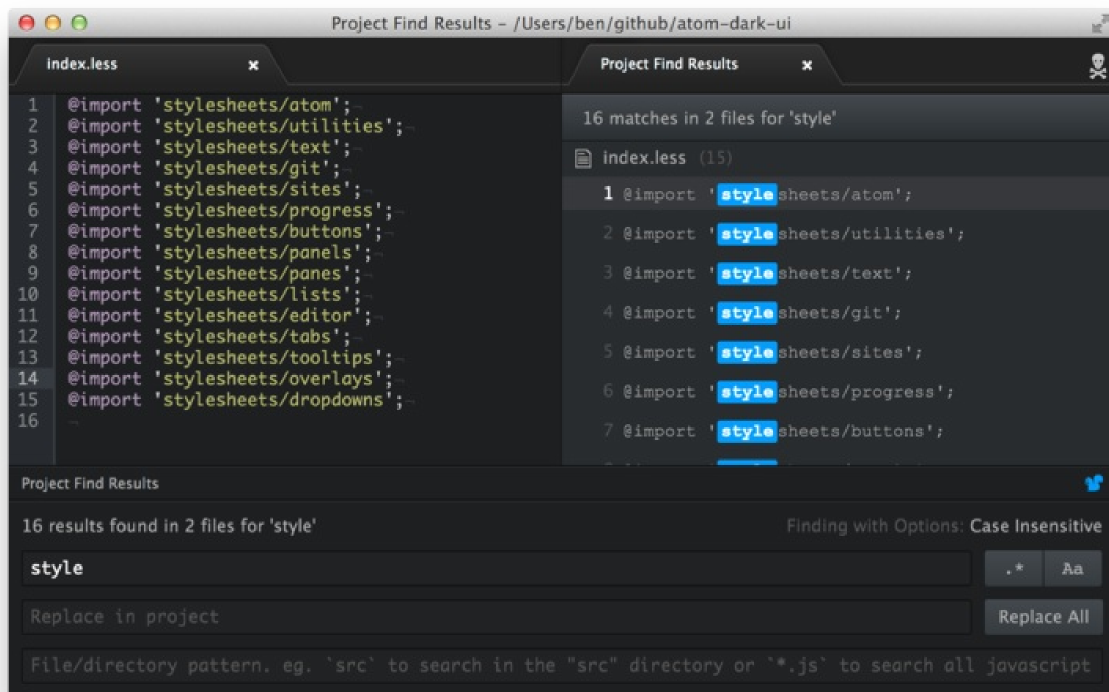
如果你执行了这些命令，你屏幕的底部会出现一个“Find and Replace”面板。



你可以按下 `cmd-F` ，输入需要搜索的字符串，并且多次按下回车（或者 `cmd-G` ，或者“Find Next”按钮）来在当前文件中搜索，循环查找当前文件中的匹配内容。“Find and Replace”也提供了一些按钮，可以设置大小写敏感，正则表达式匹配和区域搜索。

如果你在“Replace in current buffer”文本框中输入字符串，你可以将匹配到的结果替换成一个不同的字符串。例如，如果你想要把每个“Scott”字符串的实例替换成字符串“Dragon”，你可以把这些值填到两个文本框中，之后点击“Replace All”按钮来进行替换。

如果你按下 `cmd-shift-F` 来弹出面板，你也可以在整个项目中进行查找和替换。



这是一个非常棒的方法，可以找出项目中一个函数在哪里被调用，一个锚文本在哪里被连接，或者一个指定的错误拼写的位置。你可以点击匹配到的行数，来跳到它在文件中的位置。

你可以在“File/Directory pattern”文本框中输入Unix通配符，把搜索范围限制在你项目中文件的子集。当你打开了多个项目的文件夹，它还可以用于在其中一个文件夹中查找。例如，你打开了 `/path1/folder1` 和 `/path2/folder2` ，你可以输入一个以 `folder1` 开头的通配符，只在第一个文件夹中查找。

当焦点在“Find and Replac”面板上的时候，按下 `escape` 从你的工作环境中隐藏这个面板。

查找和替换功能在atom/find-and-replace包中实现，并且使用了atom/scandal包执行实际的查找。

代码段

代码段是一个非常有效的工具，可以从一个快捷方式中快速生成常用的代码语法。

这就是说，你可以输入一些类似于 `habtm` 的东西，然后按下回车键，他就会扩展为 `has_and_belongs_to_many` 。

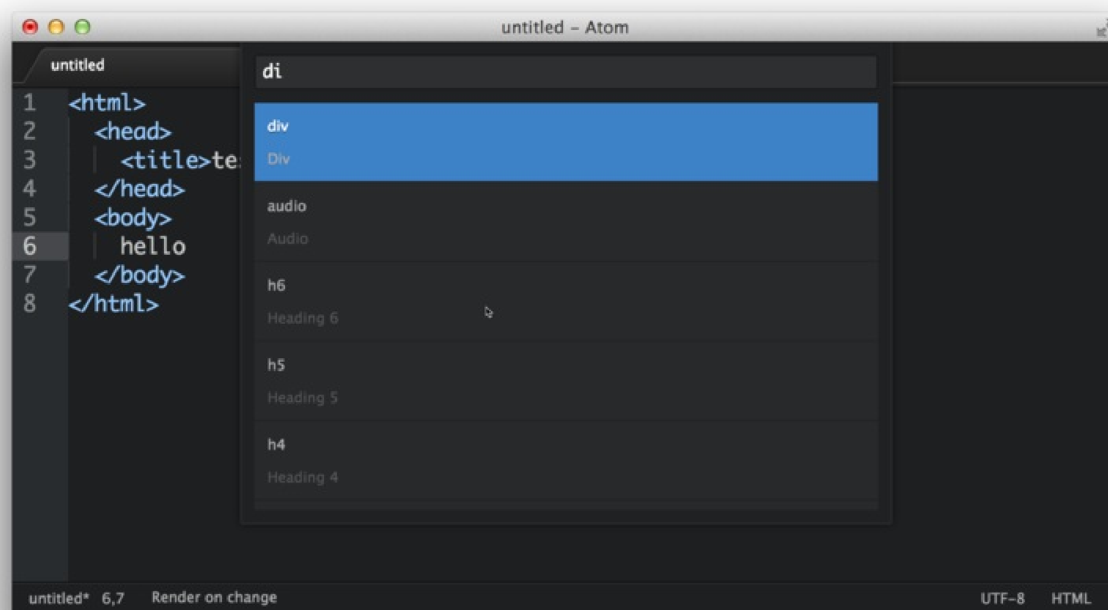
许多包自带他们自己的，具有特定模式的代码段。比如，提供了html语法高亮和语法的 `language-html` 包提供了许多代码段，来创建一些你想使用的不同HTML标签。如果你在Atom中创建一个新的HTML文件，你可以输入 `html` 然后按下 `tab` ，它会扩展为：

```
<html>
  <head>
    <title></title>
  </head>
  <body>

  </body>
</html>
```

同时它会把光标放在 `title` 标签的中间，以便你立即开始填充这个标签。许多代码段具有多个焦点位置，你可以按下 `tab` 在他们之间切换 —— 比如，在这个HTML代码段之中，你填充完标题标签之后，可以按下 `tab` 键，然后光标就会移动到body标签之间。

要查看当前打开文件拥有的所有代码段，你可以按下 `alt-shift-S` 。



你也可以在选择输入框中输入内容，来使用模糊搜索过滤这个列表。选择其中一个之后会执行光标所在的代码段（或者多个光标所在的代码段）。

创建你自己的代码段

所以说这样太爽了。但是，如果语言包中没有包含一些东西，或者你的代码中要编写一些自定义的东西，那会怎么样呢？很幸运的是，你可以非常便利地添加自己的代码段。

在你 `~/.atom` 目录下的 `snippets.cson` 文件，存放了你的所有自定义的代码段，他们会在Atom运行时加载。但是，你也可以通过 `Atom > Open Your Snippets` 菜单，轻易打开这个文件。

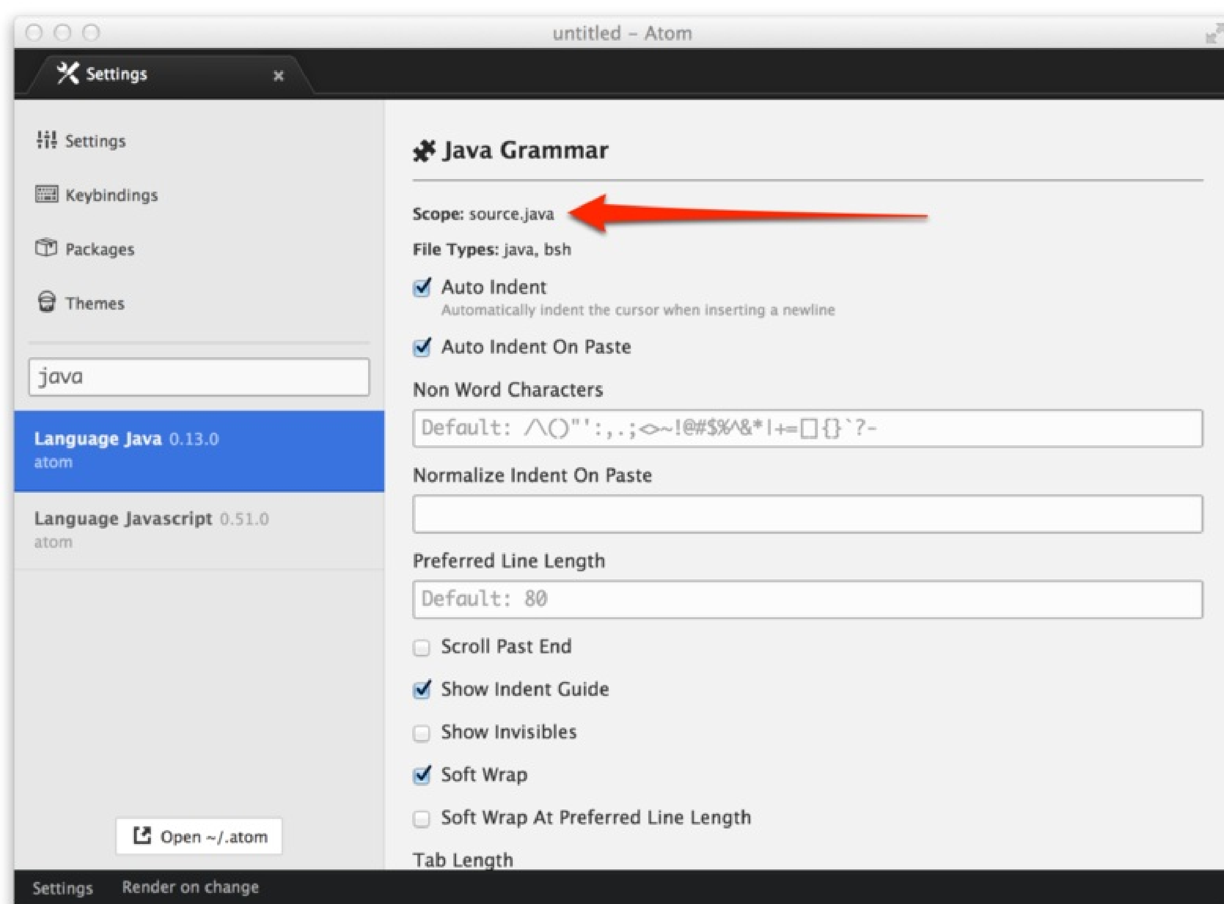
代码段的格式

现在让我们看一看如何编写代码段，基本的代码段格式像这个样子：

```
'.source.js':
  'console.log':
    'prefix': 'log'
    'body': 'console.log(${1:"crash"});$2'
```

最外面的键是选择器，即在哪里会加载代码段。决定它应该是什么的最简单的方法，是访问你想要添加代码段的语言的语言包，并找到"Scope"字符串。

例如，你想要添加在Java文件中工作的代码段，我们应该先在我们的设置视图中寻找 `language-java` 包，然后我们看到了Scope是"source.java"，代码段最顶层的键就应该是它前面加上一个点（就像CSS选择器那样）。



下一层的键是代码段的名字，用于在代码段菜单中，以一个更具可读性的方式来描述代码段。通常来说，这里最好使用对人来说具有可读性的字符串。

在每个代码段的名字下面是 `prefix` ，用于触发代码段，以及 `body` ，当代码段被触发后用于插入。

每个后面带有数字的 `$` 是tab的停止位置。在代码段被触发之后，通过按下 `tab` 键来遍历它们。

上面的例子向Javascript文件添加了 `log` 代码段，它会被扩展为：

```
console.log("crash");
```

其中的"crash"字符串会在开始时被选中，再次按下 `tab` 键之后，光标会移动到分号之后。

并不像CSS选择器，代码段的键每层只能重复一次。如果某一层有重复的键，只有最后的那个会被读到，详见配置CSON。

多行代码段主体

对于长一些的模板，你可以使用 `"""` 来使用多行语法。

```
'.source.js':
  'if, else if, else':
    'prefix': 'ieie'
    'body': """
      if (${1:true}) {
        $2
      } else if (${3:false}) {
        $4
      } else {
        $5
      }
    """
```

像你可能期待的那样，这是一个创建代码段的代码段。如果你打开一个代码段文件，输入 `snip` 之后按下 `tab` ，会将以下内容插入到文件中：

```
'.source.js':
  'Snippet Name':
    'prefix': 'hello'
    'body': 'Hello World!'
```

砰的一下，就把那个东西填充了，然后得到了一个代码段。只要你保存了文件，Atom就会重新加载它，你也就能立即使用它了。

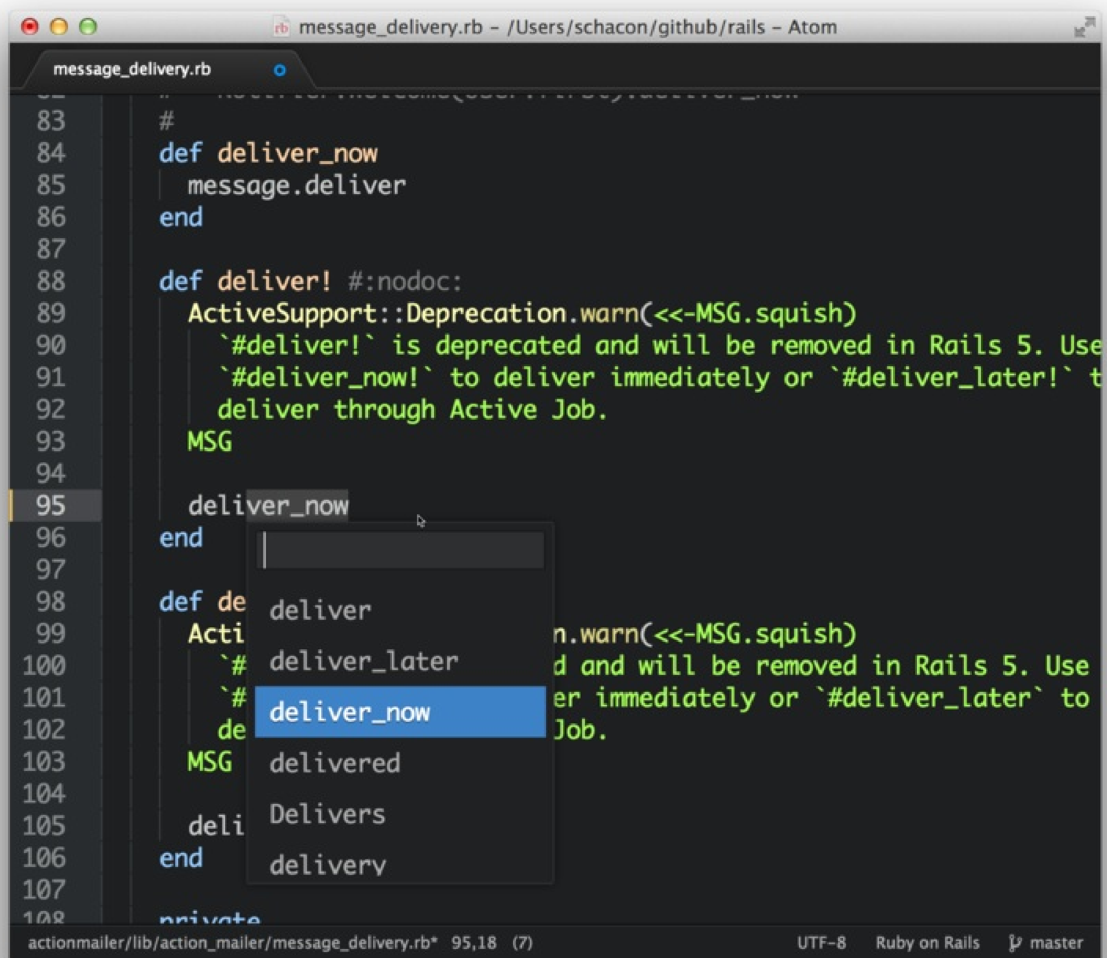
代码段功能在atom/snippets包中实现。

更多例子请见language-html中的代码段，和language-javascript包。

自动补全

如果你仍旧希望节约一些打字时间，Atom自带简单的自动补全功能。

通过使用 `ctrl-space` ，自动补全工具可以让你看到并插入可选的完整单词。



通常，自动补全工具会浏览当前打开的整个文档，寻找匹配你开始打出来的单词。

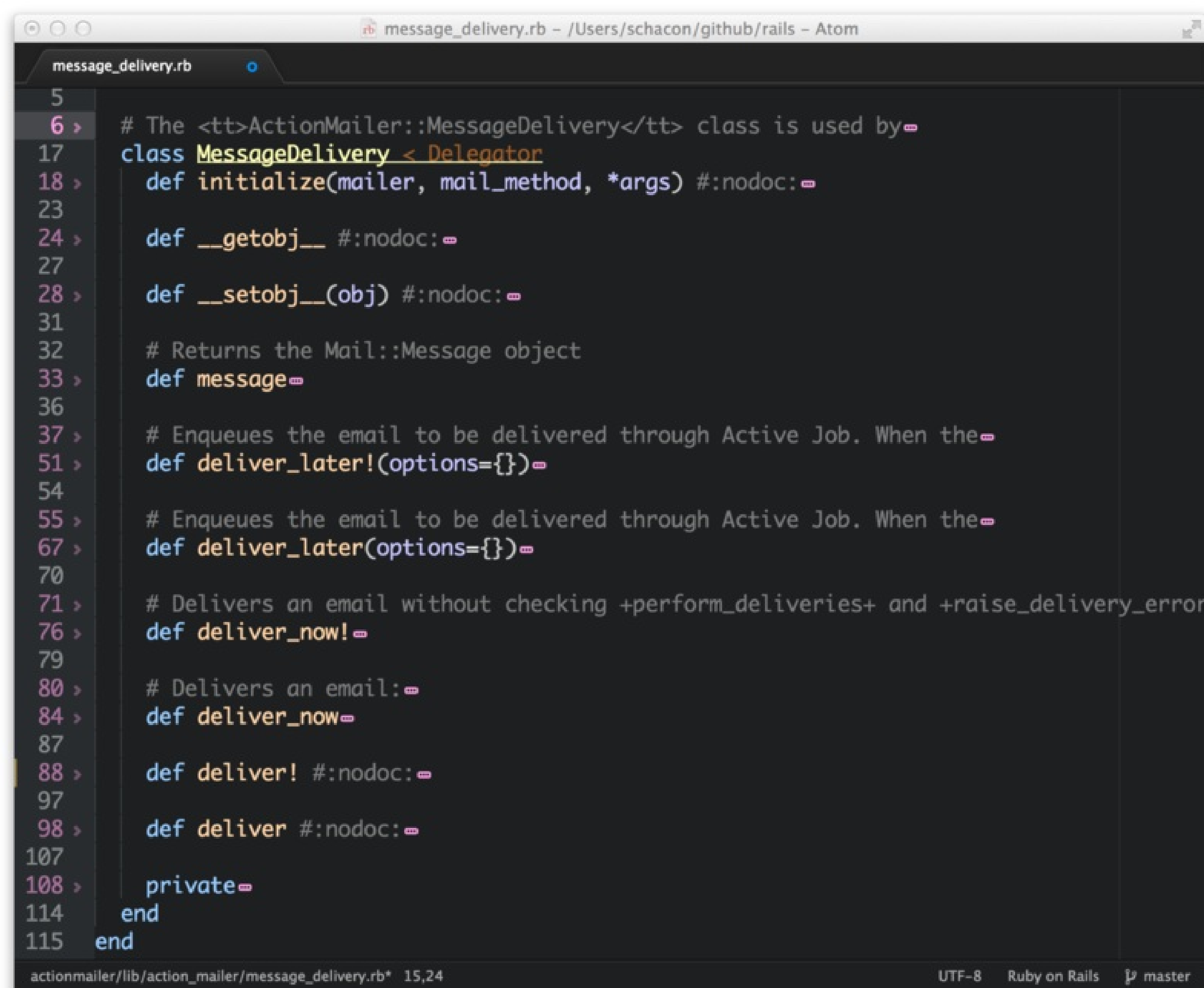
如果你想要更多选项，在设置面板的Autocomplete包中，你可以设置为在所有你打开的缓冲区中寻找字符串，而不仅仅是当前文件。

自动补全功能在atom/autocomplete包中实现。

折叠

如果你仅仅希望看到你所处理的代码文件的结构概览，折叠会是个非常有用的工具。折叠可以隐藏像函数和循环这样的代码块，来简化你屏幕上显示的东西。

当你把鼠标移到数字栏上，你就可以点击显示的箭头来折叠代码段。你也可以使用快捷键 `alt-cmd-[` 和 `alt-cmd-]` 来折叠和展开代码段。

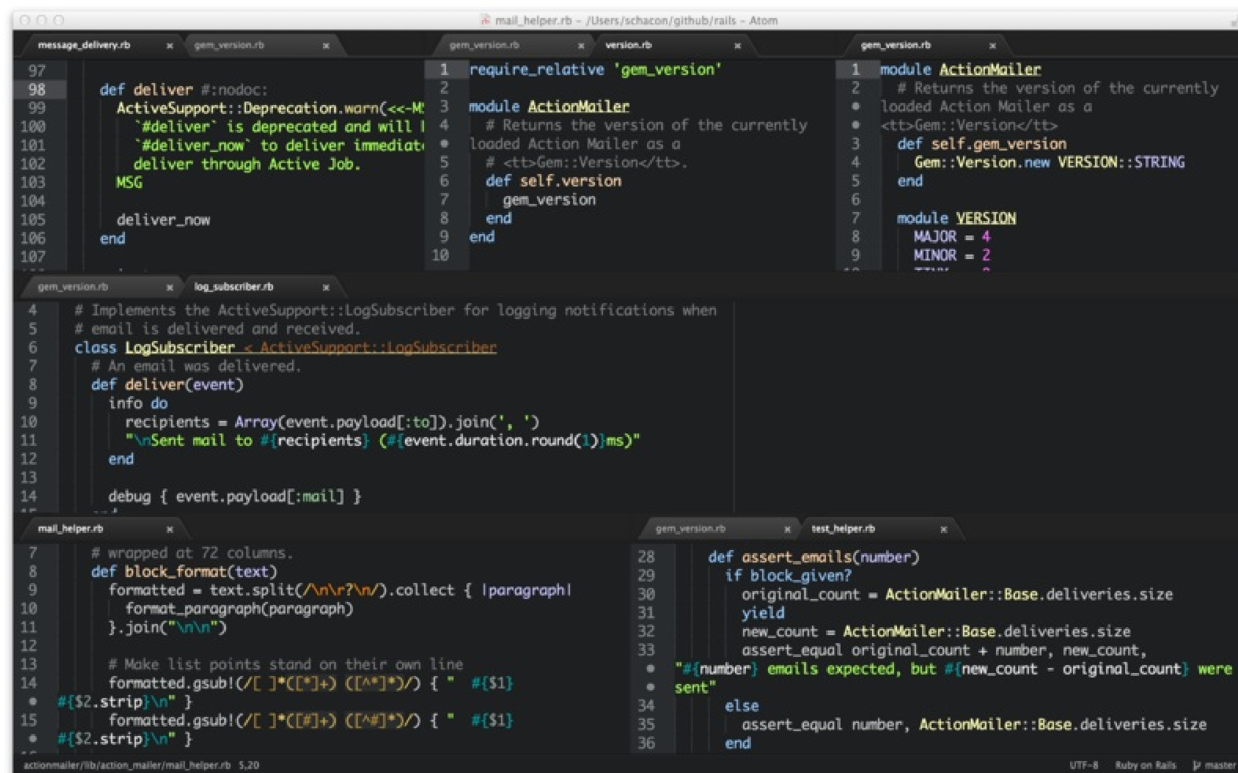


使用 `alt-cmd-shift-{` 来折叠所有代码段，使用 `alt-cmd-shift-}` 来展开所有代码段。你也可以使用 `cmd-k cmd-N` 来指定折叠的缩进级别，其中N是缩进深度。

最后，你可以折叠你代码或文本的任意一部分，通过按下 `ctrl-alt-cmd-F` ，或者在命令面板中选择“Fold Selection”。

面板

你可以通过使用 `cmd-k arrow` 快捷键来横向或者纵向分割编辑器的面板，其中“arrow”是要分割的方向。面板被分割后，你可以使用 `cmd-k cmd-arrow` 快捷键在它们之间移动焦点，其中“arrow”是焦点要移动的方向。



每个面板都有它自己的“条目”或文件，它们由标签页来表示。你可以通过拖动文件，并把它放到想要放进去的面板中，来在面板之间移动文件。

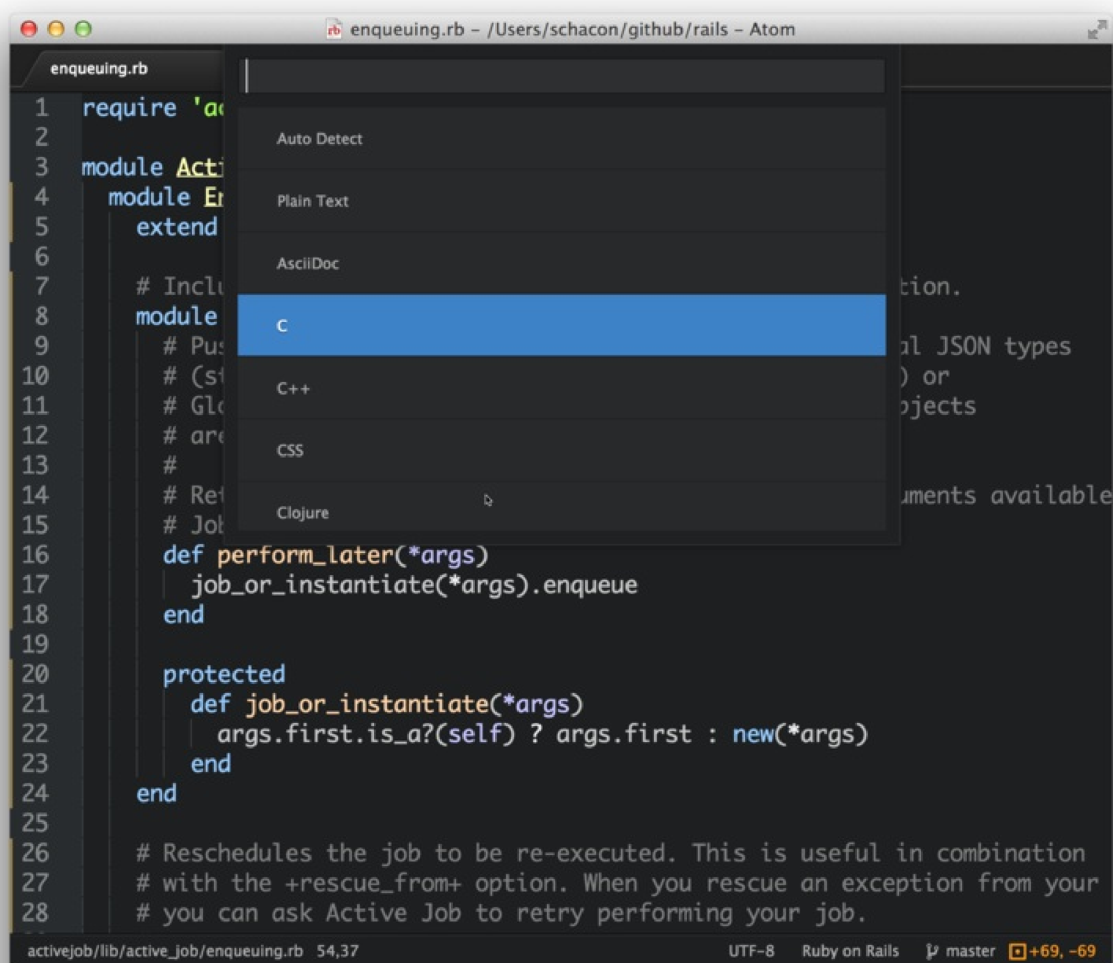
要关闭一个面板，按下 `cmd-w` 来关闭它的所有编辑器，然后再按下 `cmd-w` 几次来关闭面板。你可以在设置视图中，将面板设置为没有东西的时候自动关闭。

语法

一个缓冲区中的“语法”，是Atom所认为的，文件内容的语言类型。语法类型可以是Java或者Markdown。当我们在“Snippets”中创建代码段的时候，我们已经看到过它了。

如果你加载了一个文件，Atom会做一些工作来试图识别出文件的类型。大部分情况，Atom通过查看文件的扩展名（ `.md` 通常为一个Markdown文件，等等）来完成。然而有时扩展名难以判断，它会检查内容来进行识别。

如果你加载了一个文件，并且Atom不能判断文件的语法，它会默认为纯文本（Plain Text），这是最简单的类型。如果它把文件默认为纯文本，或者弄错了文件类型，再或者由于一些原因你想修改文件的活动语法，你可以按下 `ctrl-shift-L` 下拉语法选择工具。



一旦你手动修改了一个文件的语法，Atom会记住它，直到你将语法设置回“自动检查”，或者手动选择一个不同的语法。

语法选择工具的功能在atom/grammar-selector包中实现。

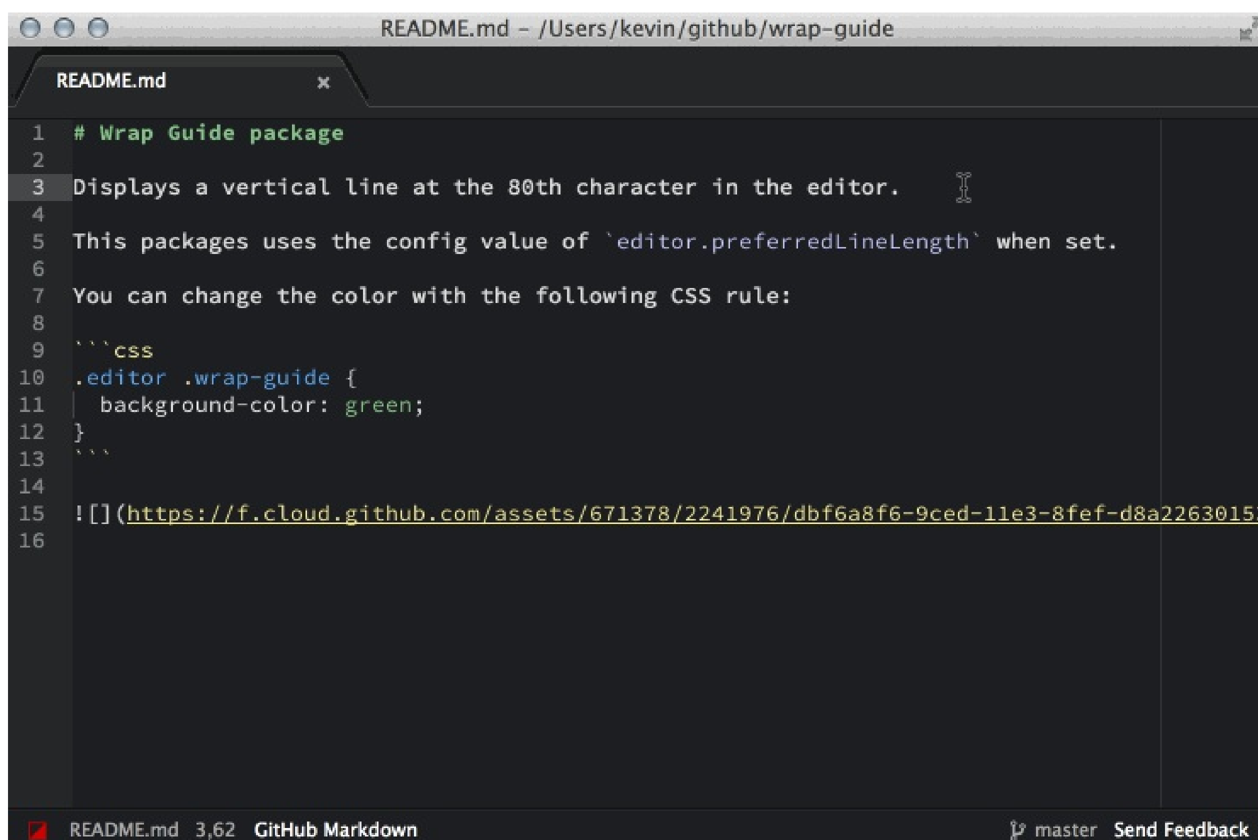
Atom中的版本控制

对于任何项目来说，版本控制都是很重要的一个方面。Atom集成了一些基本的Git和Github功能。

检出（checkout）HEAD中的版本

`cmd-alt-z` 快捷键检出当前文件在HEAD中的版本。

这是一个快捷的方法，来撤销所有你保存的或者阶段性的修改，并且把你的文件还原到HEAD中（最后提交）的版本。这从本质上相当于使用命令行在 `path` 中执行 `git checkout HEAD -- <path>` 或者 `git reset HEAD -- <path>` 命令。

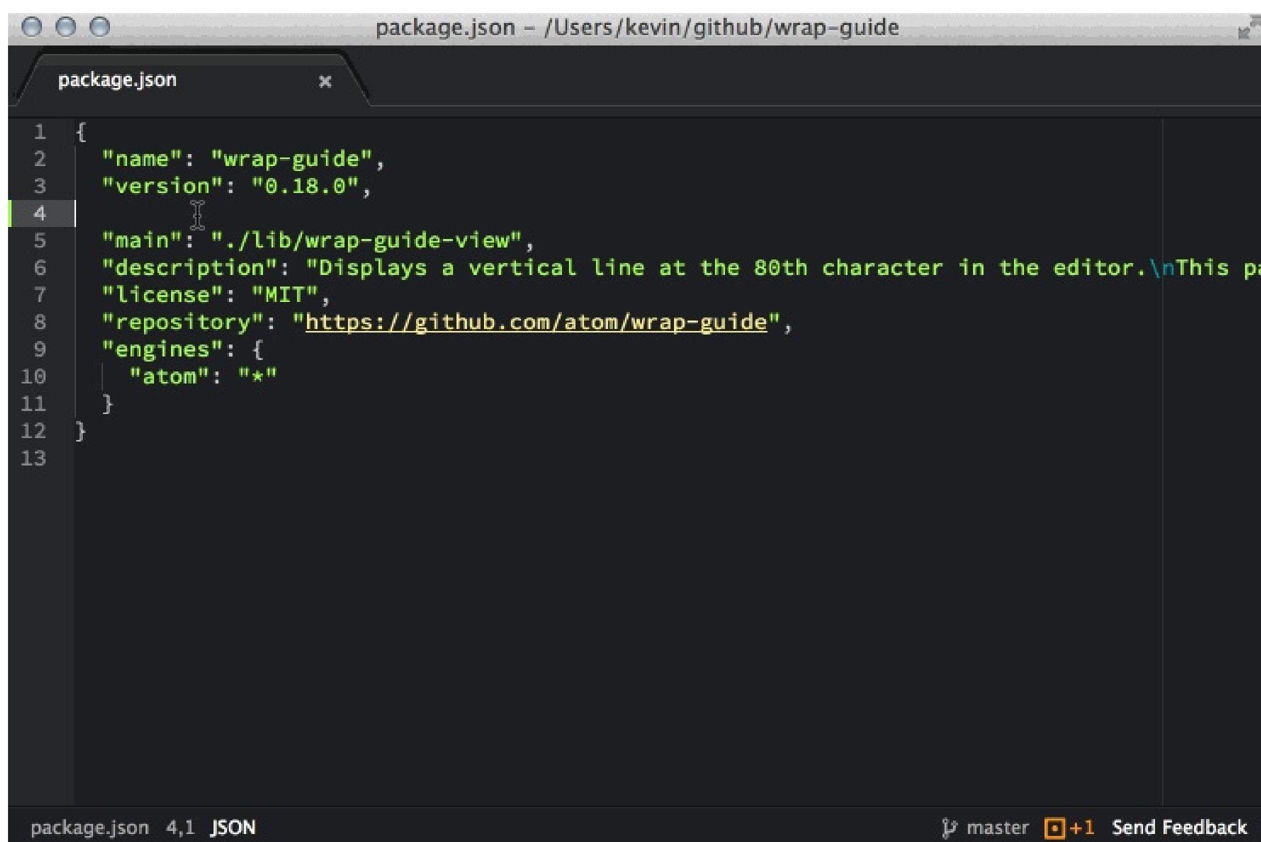


这个命令会保存到撤销栈，所以稍后你可以使用 `cmd-z` 来恢复之前的内容。

Git状态（status）列表

Atom带有模糊查找的包，提供了 `cmd-T` 快捷键来快速打开项目中的文件，以及 `cmd-B` 快捷键来跳到任何已打开的编辑器。

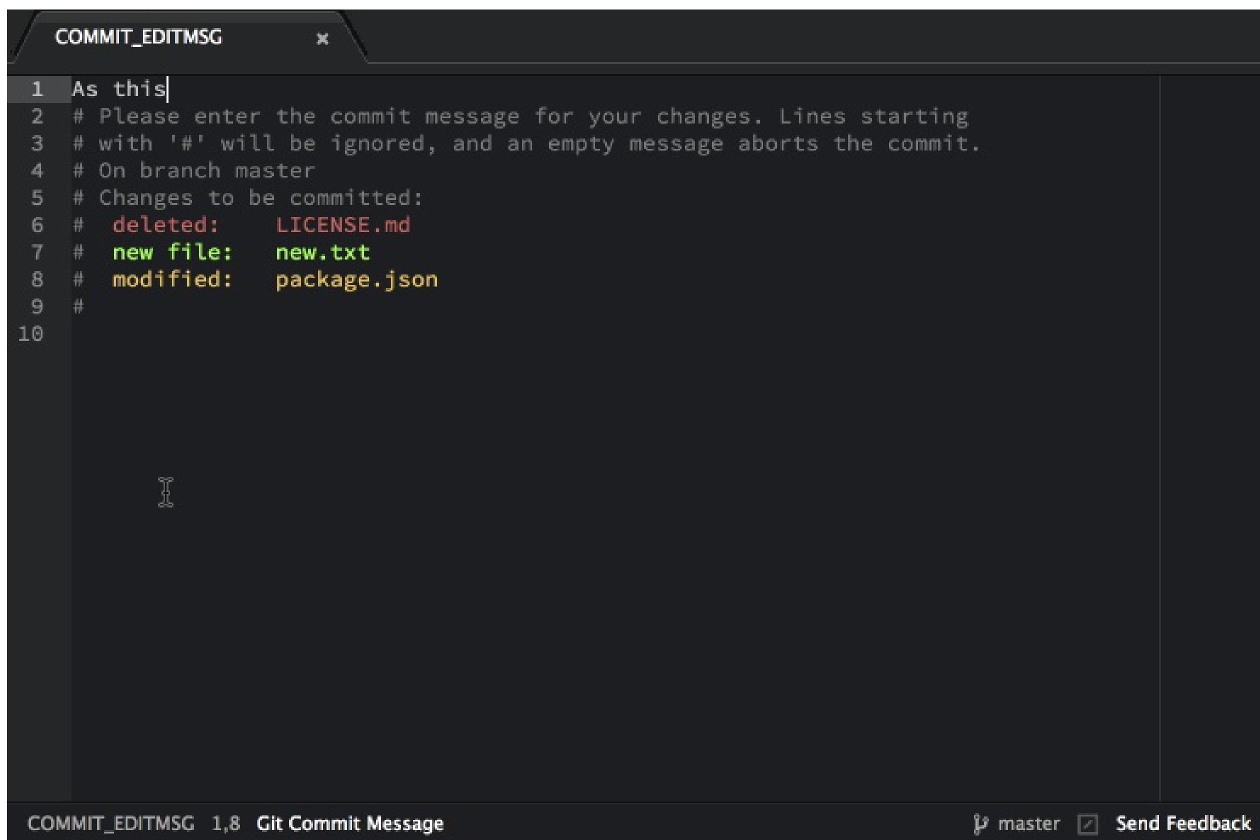
这个包也提供了 `cmd-shift-B` 快捷键，用来显示所有未跟踪和已修改的文件列表。如果你运行 `git status` ，你在命令行中会看到相同的文件。



每个文件的右边会出现一个小图标，让你知道它是未跟踪的还是已修改的。

提交（commit）编辑器

Atom可以用作你的Git提交（commit）编辑器，并自带git语法包（language-git），它添加了语法高亮来编辑提交（commit）、合并（merge）和rebase消息。



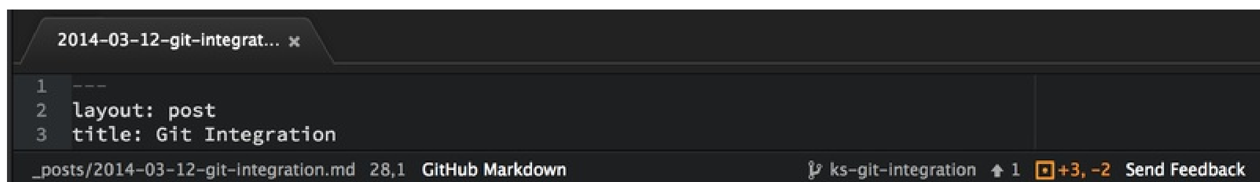
你可以使用以下命令来设置Atom为你的Git提交编辑器。

```
$ git config --global core.editor "atom --wait"
```

language-git包会通过给提交消息的第一行加上颜色，来提醒你缩短它，当它超过50和65个字符的时候。

状态栏的图标

status-bar包带有一些Git标识，用于显示在状态栏的右边。



当前检出的分支名称，会和当前分支在上游（upstream）分支之前或之后的提交数量一起显示。

如果当前文件未跟踪、已修改或者被忽略，就会添加一个标识。最后一次提交以来的添加和删除的行数也会显示。

行间差异

引入的git-diff包在行号旁边为添加、修改和删除的行着色。

```
LICENSE.md ×
1  Copyright (c) 2014 GitHub Inc.
2
3  ADDED!
4  Permission is hereby granted, free of charge, to any person obtaining
5  a copy of this software and associated documentation files (the
6  "Software"), to deal in the Software without restriction, including
7  without limitation the rights to use, copy, modify, merge, publish,
8  EDITED!
9  permit persons to whom the Software is furnished to do so, subject to
10 the following conditions:
11
12
13 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
14 EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
15 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
16 NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
17 LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
18 OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
19 WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
20
```

这个包也添加了 `alt-g down` 和 `alt-g up` 快捷键，允许你在当前编辑器中把光标移动到上一个或下一个不同的代码块。

在Github上打开

如果你处理的项目存放在Github上，你可以使用许多方便的集成功能。这些命令的大多数都作用于你当前查看的文件，并在Github上打开它的视图 —— 例如，当前文件的修改历史（blame）或者提交历史（commit history）。

`alt-G O`

在Github上打开文件。

`alt-G B`

在Github上打开文件的修改历史。

`alt-G H`

在Github上打开文件的提交历史。

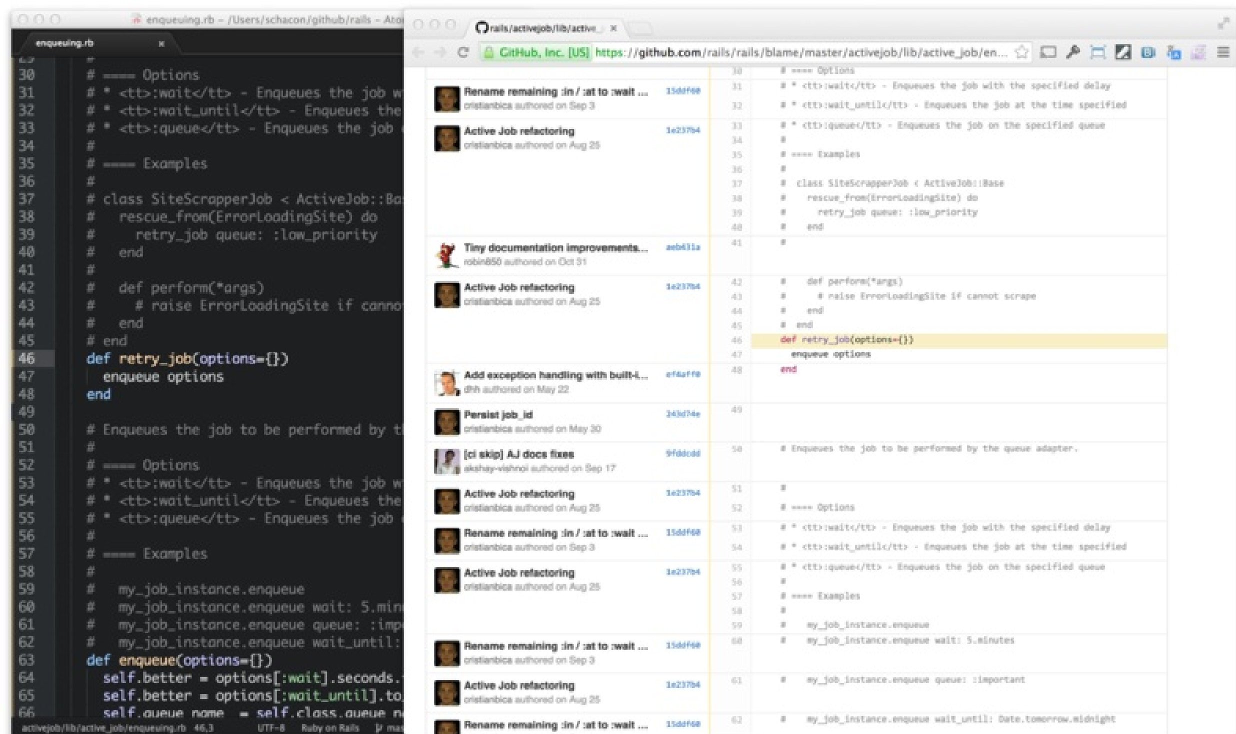
`alt-G C`

复制当前文件在Github上的链接。

`alt-G R`

在Github上进行分支比较。

分支比较只是简单地向你展示那些在本地的当前工作分支上存在，并且在主分支上没有的提交。



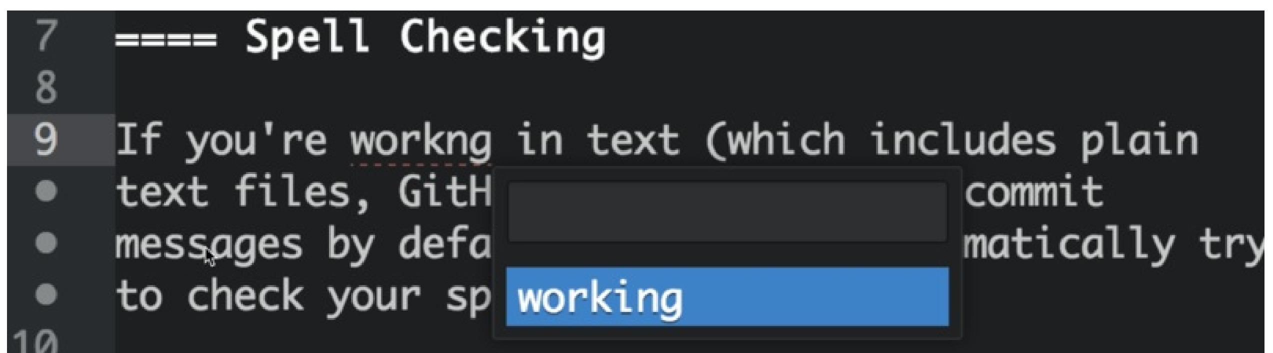
在Atom中写作

虽然Atom通常可能用来编写软件的代码，但是它还可以用来高效地编写文章。这通常采用一些标记语言，比如说Markdown和Asciidoc（也就是英文手册所用的格式）来完成。下面我们会很快浏览一遍Atom提供给你用来写文章的一些工具。

拼写检查

如果你在处理文本（通常包括纯文本文件，Github Markdown文件和Github提交信息），Atom会自动尝试去检查你的拼写。

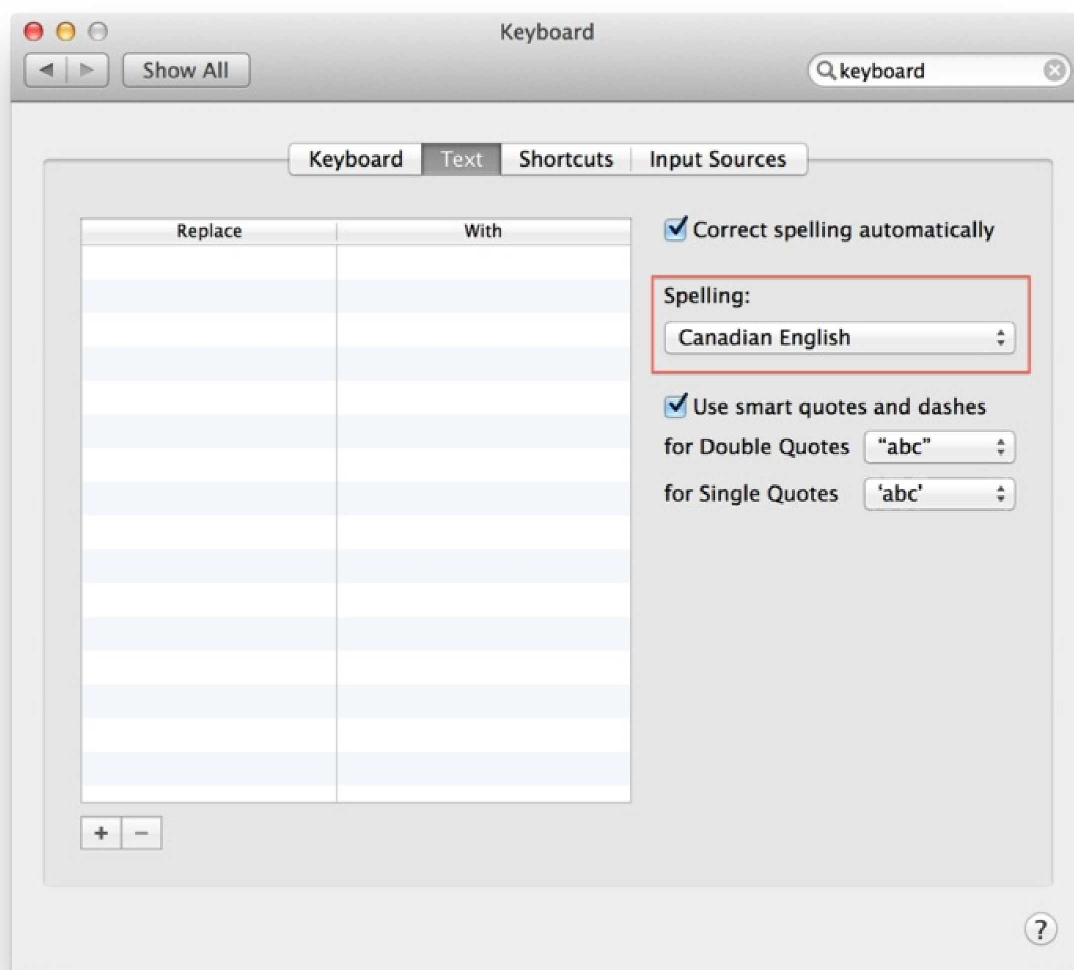
任何拼写错误的单词都会高亮显示（通常以单词下方的点状红色下划线），你可以按下 `cmd-:` 来拉出菜单查看可能的正确选项（或者从右键弹出的菜单中，或命令面板中选择“Correct Spelling”）。



要向Atom尝试检查拼写的列表中添加文件类型，在你的设置视图中访问拼写检查包的设置，然后添加你希望检查的任何语法。

需要检查的默认语法是“text.plain, source.gfm, text.git-commit”，但是你可以添加类似“source.asciidoc”的东西，如果你希望也检查这种类似的文件。

Atom拼写检查工具使用系统的字典，所以如果你希望在另一种语言或者区域中检查拼写，你可以很容易修改它。



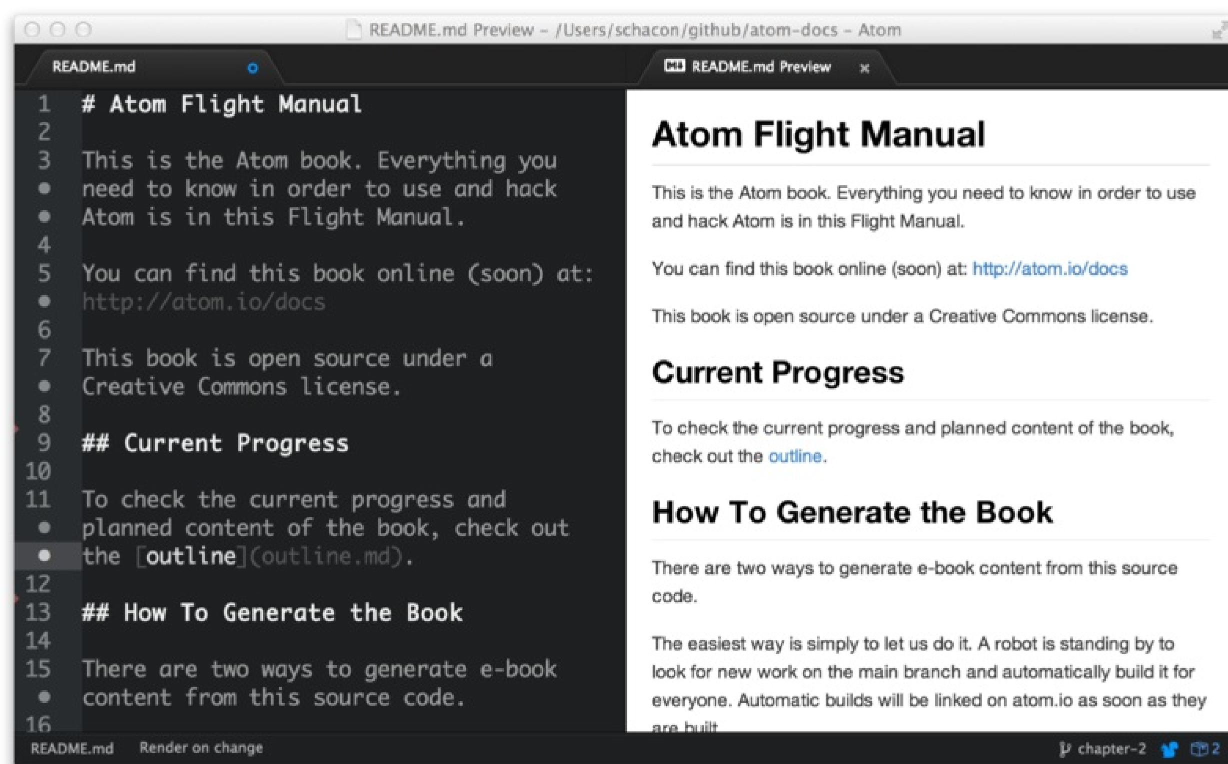
拼写检查功能在atom/spell-check包中实现。

预览

使用Markdown写文章的时候，从渲染后的内容的样子中得到一个想法还是很有用的。Atom中默认带有Markdown预览插件。

`ctrl-shift-M`

为Markdown开启预览模式。



在你编辑文本时，预览也会默认自动更新。这样你在打字时检查语法就变得容易了。

你也可以从预览面板中，复制任何渲染后的HTML到系统剪贴板中。这个操作没有任何快捷键，但是你可以在命令面板中通过搜索“Markdown Preview Copy HTML”来找到它。

Markdown预览在atom/markdown-preview包中实现。

代码段

有很多好用的代码段是为快速编写Markdown准备的。

如果你输入 `img` 之后按下 `tab` ，你会得到像 `![]()` 这样的Markdown格式的图片代码。如果你输入 `table` 之后按下 `tab` ，你会得到一个非常棒的用于填充的示例表格。

```
| Header One     | Header Two     |
| :------------- | :------------- |
| Item One       | Item Two       |
```

虽然用于Markdown的代码段不多（ `b` 粗体， `i` 斜体， `code` 代码块，等等），它们会节省你用于寻找模糊的语法的时间。另外，你可以按下 `alt-shift-S` ，来寻找当前文件类型可用的代码段列表。

基本的自定义

在我们感受到Atom中所有东西的便利之后，让我们看看如何改进它。可能有一些快捷键你经常使用但是感觉很别扭，或者一些颜色不是十分适合你。Atom具有惊人的灵活性，所以让我们对它做一些力所能及的简单调整。

使用CSON来配置

所有Atom的配置文件（除了你的样式表和初始脚本）全部用CSON编写，全称是CoffeeScript Object Notation。就像JSON（JavaScript Object Notation）的名字一样，CSON是一个储存结构化数据的文本格式，表现为由键值对组成的简单对象的形式。

```
key:
  key: value
  key: value
  key: [value, value]
```

对象是CSON的基石，由缩进（像上面的文件那样）或者花括号（ `{}` ）描述。一个键的值可以是字符串、数字、对象、布尔值、 `null` 或者上述数据类型的一个数组。

不像CSS的选择器，CSON的键在每个对象中只能重复一次。如果存在重复的键，最后一次出现的那个会覆盖其他所有同名的键。在Atom配置文件中也是如此。

避免这种情况：

```
# DON'T DO THIS
'.source.js':
  'console.log':
    'prefix': 'log'
    'body': 'console.log(${1:"crash"});$2'

# Only this snippet will be loaded
'.source.js':
  'console.error':
    'prefix': 'error'
    'body': 'console.error(${1:"crash"});$2'
```

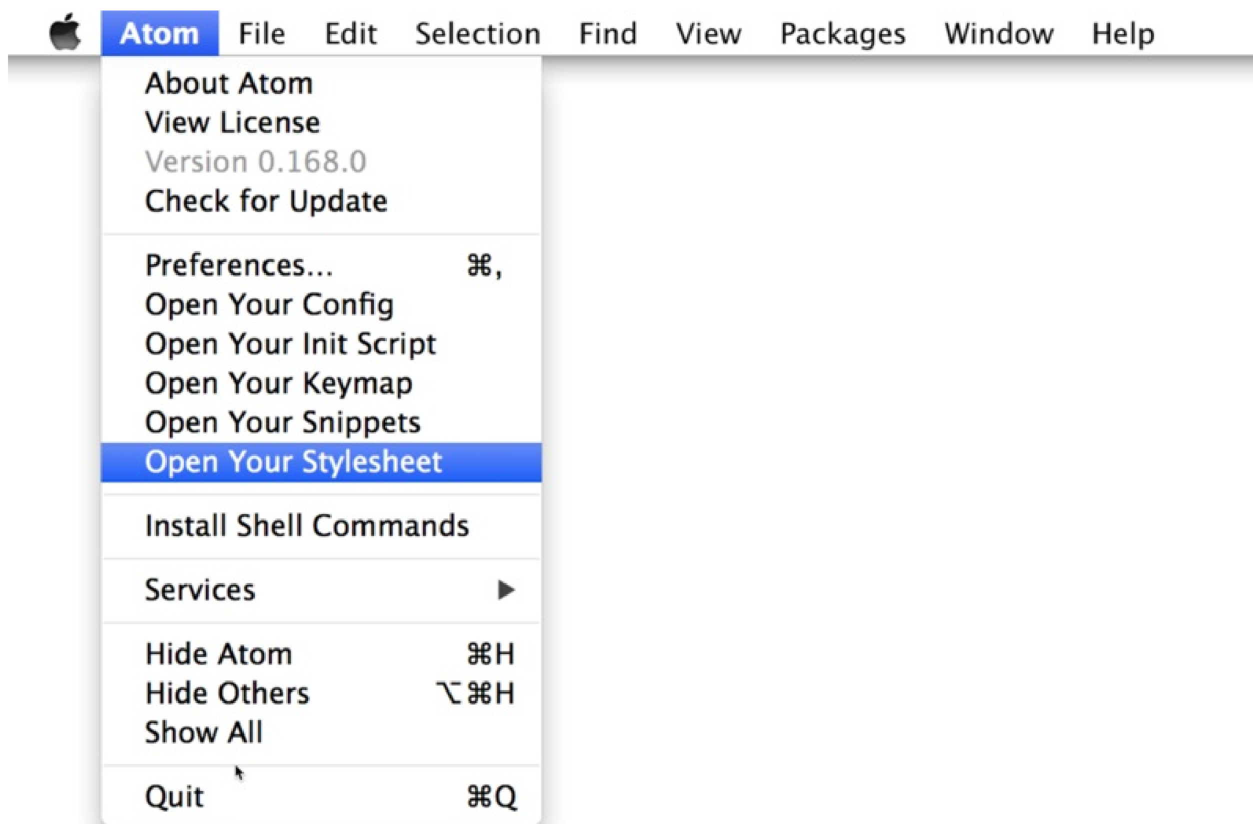
而是要写成这样：

```
# DO THIS: Both of these will be loaded
'.source.js':
  'console.log':
    'prefix': 'log'
    'body': 'console.log(${1:"crash"});$2'
  'console.error':
    'prefix': 'error'
    'body': 'console.error(${1:"crash"});$2'
```

样式调整

如果你只是对个人样式做一些应急的修改，而不打算发布整个主题，你可以在你的 `~/.atom` 目录的 `styles.less` 文件中添加样式。

你可以在编辑器中从 `Atom > Open Your Stylesheet` 菜单打开这个文件。

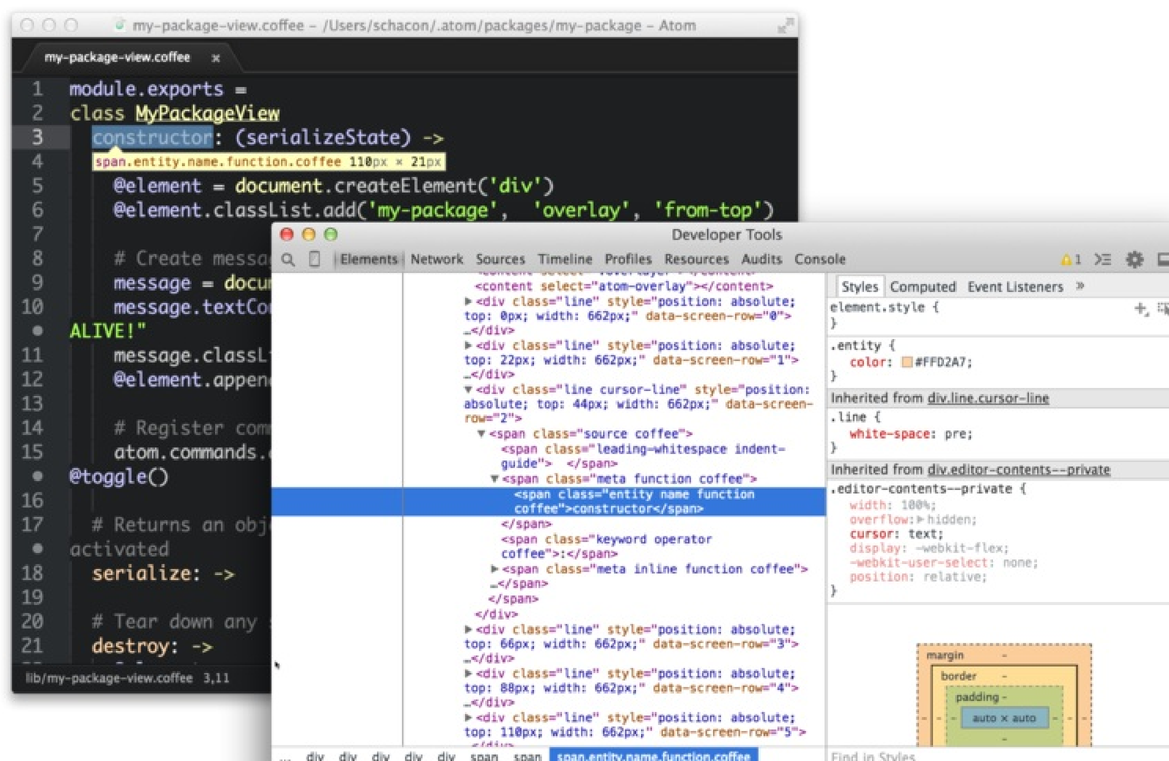


例如，要修改光标的颜色，你可以将一下规则添加到你的 `~/.atom/styles.less` 文件中：

```
atom-text-editor::shadow .cursor {
  border-color: pink;
}
```

了解都有哪些class可用的最简单方式，是通过开发者工具手动查看DOM。我们将在下一章详细介绍这个工具，现在先简单来看一下。

你可以通过按下 `alt-cmd-I` 来打开开发者工具，然后会弹出一个Chrome开发者工具面板。



你可以轻易查看到当前编辑器的所有元素。如果你想更新一些东西的样式，你需要先知道它拥有哪个class，然后再你的样式文件中添加一条Less规则。

如果你不熟悉Less，它是一个让CSS变得更简单的CSS预处理器，你可以访问lesscss.org来了解关于它的更多信息。如果你更愿意使用CSS，这个文件也可以命名为styles.css来包含CSS。

自定义快捷键

Atom从你 `~/.atom` 目录中的 `config.cson` 文件中加载配置，它含有CoffeeScript格式的JSON，也就是CSON：

```
'core':
  'excludeVcsIgnoredPaths': true
'editor':
  'fontSize': 18
```

配置本身以包名分组，或者一两个核心的命名空间，比如 `core` 和 `editor` 。

你可以从 `Atom > Open Your Config` 菜单在编辑器中打开它。

快捷键配置参考

- `core`

- `disabledPackages` ：被禁用的包名的一个列表
- `excludeVcsIgnoredPaths` ：不要加载 `.gitignore` 指定的文件
- `ignoredNames` ：在Atom中要忽略的文件名
- `projectHome` ：假定项目被存放的目录
- `themes` ：要加载的主题名称的数组，按照层叠顺序

- `editor`

  - `autoIndent` ：开启或关闭基本的自动缩进（默认为true）
  - `nonWordCharacters` ：一个非单词字符的字符串，来指定单词边界
  - `fontSize` ：编辑器中的字体大小
  - `fontFamily`：编辑器中的字体类型
  - `invisibles`：一个Atom用来渲染空白字符的哈希表。键是空白字符的类型，值是被渲染成的字符（使用false来屏蔽单个的空白字符）
    - `tab` ：硬tab字符
    - `cr` ：回车（Carriage return，微软风格的行末尾）
    - `eol：字符 \n``
    - `space` ：在开头或末尾的空格字符
  - `preferredLineLength` ：设定一行的长度（默认为80）
  - `showInvisibles` ：是否将不可见字符渲染为占位符（默认为false）
  - `showIndentGuide` ：是否在编辑器中显示缩进标识
  - `showLineNumbers` ：显示或者隐藏行号
  - `softWrap` ：开启或关闭编辑器中的软换行
  - `softWrapAtPreferredLineLength`：开启或关闭在 `preferredLineLength` 处软换行
  - `tabLength` ：tab字符所占空格字符的宽度（默认为2）

- `fuzzyFinder`
  - `ignoredNames` ：只在模糊查找中忽略的文件名
- `whitespace`
  - `ensureSingleTrailingNewline` ：是否将文件末尾的多个换行减少为一个
  - `removeTrailingWhitespace` ：开启或关闭清除行尾的空白字符（默认为true）
- `wrap-guide`
  - `columns` ：带有 `pattern` 和 `column` 键的数组，用来将当前编辑器的目录匹配到列中的位置

语言特定配置

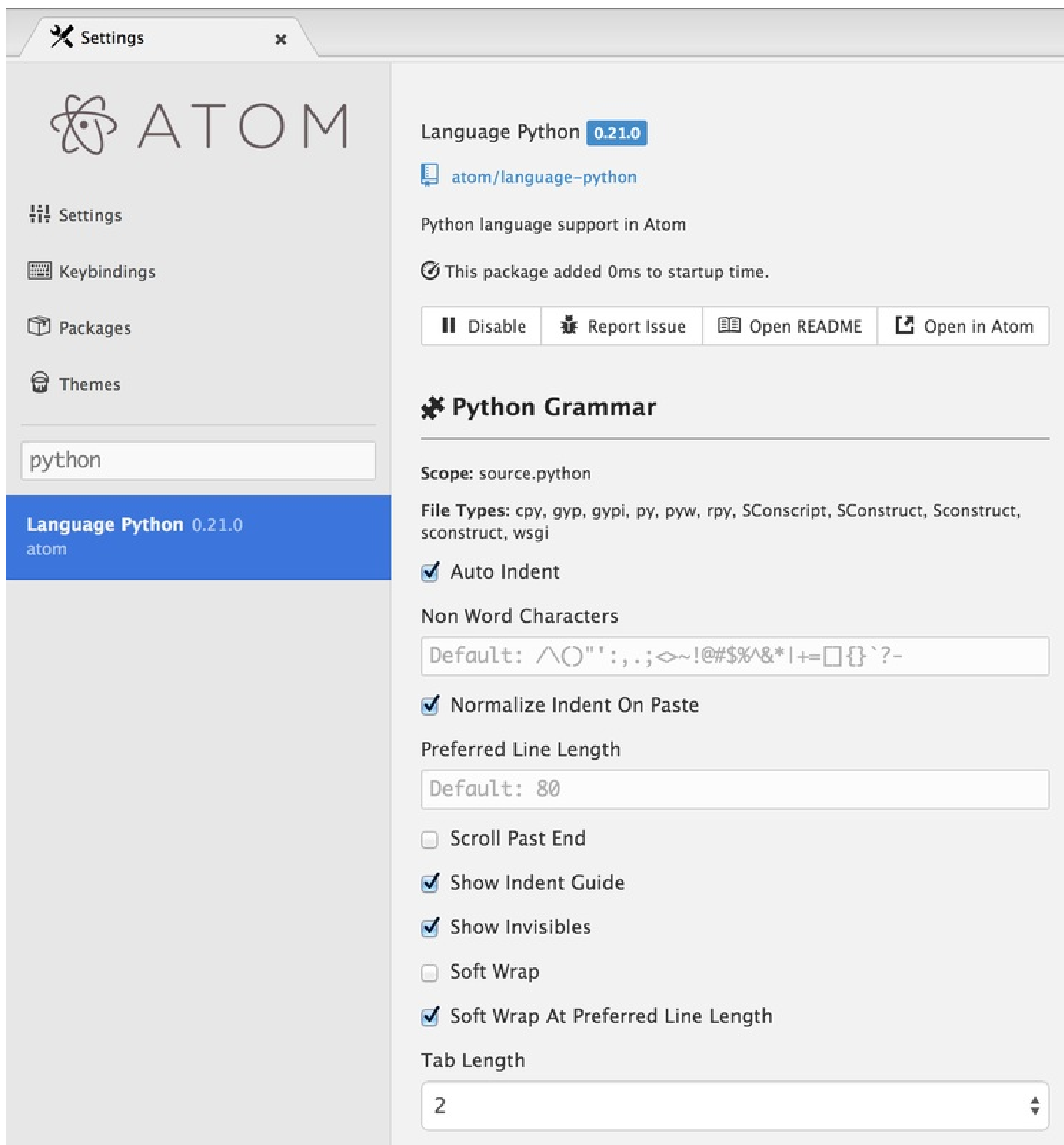
你也可以为不同的文件类型设置几种不同的配置。例如，你可能希望Atom在Markdown文件中软换行，在ruby文件中将tab显示为两个空格的宽度，在python文件中显示为4个空格的宽度。

下面是作用于语言的几种选项，这是它们的列表：

```
editor.tabLength
editor.softWrap
editor.softWrapAtPreferredLineLength
editor.preferredLineLength
editor.scrollPastEnd
editor.showInvisibles
editor.showIndentGuide
editor.nonWordCharacters
editor.invisibles
editor.autoIndent
editor.normalizeIndentOnPaste
```

设置视图中的语言特定配置

你可以在设置视图中的每个语言的包中，编辑这些配置。只要在左边的面板搜索你选择的语言，选择它，并且编辑它就好了。



配置文件中的语言特定配置

你也可以直接编辑实际的配置文件。通过在命令面板中输入“open config”并按下回车来打开配置文件。

全局设置在 `global` 键的下面。每种语言都有他们自己的顶级键，这个键就是这种语言的作用域。语言特定的设置会覆盖全局域的任何设置。

```
'global': # all languages unless overridden
  'editor':
    'softWrap': false
    'tabLength': 8

'.source.gfm': # markdown overrides
  'editor':
    'softWrap': true

'.source.ruby': # ruby overrides
  'editor':
    'tabLength': 2

'.source.python': # python overrides
  'editor':
    'tabLength': 4
```

查找语言作用域的名字

为了有效地编写这种覆盖的设置，你需要知道语言的作用域名称。我们已经在“代码段格式”一节中，为了编写代码段而做过一遍了，现在我们简单地重复一下。

作于域的名称显示在设置视图中的每个语言的包中。在左边的面板中寻找你选择的语言，选择它，然后你会在语言名称的标题下面看到作用域名称：

小结

到目前为止，你应该是一个Atom高级用户了。你应该能够像一个行家那样浏览和处理文本和文件。你也应该能够从里到外定制Atom，来让它看起来和表现得和你想象中一样。

在下一章，我们打算更上一层楼：我们会看一看如何修改和增加Atom核心中的功能。我们也准备为Atom创建新的包。只要你想得到，我们就能做得到。

Hacking Atom

现在是时候来介绍在这一Hackable的编辑器中，真正“Hackable”的部分了。像我们在整个第二章中看到的那样，Atom中很大一部分都由一大堆包组成。如果你想向Atom中添加一些功能，你必须要访问和Atom核心特性相同的API和工具。从树视图、命令面板到查找替换功能，甚至Atom的绝大多数核心特性都是以包的形式实现的。

所需工具

最开始，我们假设在某种程度上，你只知道很少的事情。由于Atom完全采用web技术实现，我们必须假设你知道CoffeeScript和Less的任何事情，它们是Javascript和CSS的预处理器。

如果你不了解CoffeeScript，但是熟悉JavaScript，就应该没有太大问题。下面是一个CoffeeScript的简单示例：

```
MyPackageView = require './my-package-view'

module.exports =
  myPackageView: null

  activate: (state) ->
    @myPackageView = new MyPackageView(state.myPackageViewState)

  deactivate: ->
    @myPackageView.destroy()

  serialize: ->
    myPackageViewState: @myPackageView.serialize()
```

我们会展示一些这样的例子，而这就是这门语言的样子。

在Atom中，你能用CoffeeScript做的任何事情都可以用JavaScript来做，但是由于大多数社区都用CoffeeScript，你可能会想用它来编写你的包。这会有助于你从开源社区中获取代码，并且在许多实例中编写更简单的代码。

你可以在coffeescript.org上面复习CoffeeScript。

Less是源于CSS的简化转换，它向CSS中添加了一些有用的东西，比如变量和函数。你可以在lesscss.org上面复习Less的技巧。但是我们对Less的使用并不涉及到这本书中太复杂的部分，所以你只要懂得CSS的基本知识就够了。

初始文件

当Atom完成加载之后，它会运行你 `~/.atom` 目录中的 `init.coffee` 文件，给你一个机会来运行CoffeeScript代码来执行自定义。这个文件中的代码可以充分访问到Atom API。如果自定义的代码变得很大，考虑创建一个包，这部分会在“字数统计包”一节中介绍。

你可以在编辑器中从 `Atom > Open Your Init Script` 菜单打开 `init.coffee` 文件。这个文件也可以命名为 `init.js` 来包含JavaScript代码。

例如，如果你在设置中开启了蜂鸣提示音，你可以将以下代码添加到 `init.coffee` ，让Atom在每次加载时用蜂鸣提示音向你打招呼。

```
atom.beep()
```

由于 `init.coffee` 可以访问到Atom API，你可以使用它来实现有用处的命令，而不需要创建新的或者扩展现有的包。下面是一个使用了选择APi和剪贴板API的命令，它从被选中的文本和剪贴板内容中构建Markdown连接作为URL：

```
atom.commands.add 'atom-text-editor', 'markdown:paste-as-link', ->
  return unless editor = atom.workspace.getActiveTextEditor()

  selection = editor.getLastSelection()
  clipboardText = atom.clipboard.read()

  selection.insertText("[#{selection.getText()}](#{clipboardText})")
```

现在，重新加载Atom，并使用命令面板通过名字执行新的命令（例如“Markdown: Paste As Link”）。而且，如果你喜欢通过快捷键来触发这个命令，你可以为命令定义一个键表（keymap）。

字数统计包

（待翻译）

文本处理包

在我们写完第一个包之后，让我们看一看我们能写出来的其它包的例子。这一节会引导你创建一个简单的命令来将选中的文字替换为字符画（ascii art）。在你在单词“cool”选中的时候运行我们的命令，它会被替换为：

```
                                     o888
  ooooooo     ooooooo     ooooooo    888
888     888 888     888 888     888  888
888         888     888 888     888  888
  88ooo888    88ooo88     88ooo88   o888o
```

这个例子应该展示了如何在当前的文本缓冲区做基本的文字操作，以及如何处理选择。

最后的包在 https://github.com/atom/ascii-art 中查看。

基本的文字插入

首先按下 `cmd-shift-P` 来弹出命令面板。然后输入“generate package”并且选择“Package Generator: Generate Package”命令，就像我们在“包生成器”一节中做的那样。输入 `ascii-art` 作为包的名字。

现在让我们编辑包中的文件，来让我们的字符画包做一些有意思的事情。由于这个包并不需要任何UI，我们可以把所有视图相关的移除，所以可以放心删除 `lib/ascii-art-view.coffee` 、 `spec/ascii-art-view-spec.coffee` 和 `styles/` 。

接下来，打开 `lib/ascii-art.coffee` 并删除所有视图代码，所以它看起来像这样：

```
{CompositeDisposable} = require 'atom'

module.exports =
  subscriptions: null

  activate: ->
    @subscriptions = new CompositeDisposable
    @subscriptions.add atom.commands.add 'atom-workspace',
      'ascii-art:convert': => @convert()

  deactivate: ->
    @subscriptions.dispose()

  convert: ->
    console.log 'Convert text!'
```

创建命令

现在让我们添加一个命令。强烈建议你为你的命令取一个命名空间，使用包名后面带着一个 `:` 。所以你可以看到在代码中，我们把命令叫做 `ascii-art:convert` ，并且当它调用时会调用 `convert()` 方法。

到目前为止，它只会在控制台中记录。让我们使它向文本缓冲区插入一些字符来开始。

```
convert: ->
  if editor = atom.workspace.getActiveTextEditor()
    editor.insertText('Hello, World!')
```

就像在"字数统计"中那样，我们使用 `atom.workspace.getActiveTextEditor()` 来获取表示当前活动编辑器的对象。如果 `convert()` 方法在没有编辑器获取焦点时调用，它会简单地返回一个空白的字符串，所以我们可以跳过下一行。

接下来我们使用 `insertText()` 方法，向当前的文本编辑器插入一个字符串。无论光标当前在编辑器的哪里，都会在光标处插入文本。如果有文本被选中，会把选中文本替换成"Hello, World!"文本。

重新加载包

在我们能够触发 `ascii-art:convert` 之前，我们需要通过重新加载窗口，来加载我们的包的最新代码。从命令面板或按下 `ctrl-alt-cmd-l` 来运行"Window: Reload"命令。

触发命令

现在可以打开命令面板并搜索"Ascii Art: Convert"命令了。但是根本找不到。要修正它，打开 `package.json` 并找到 `activationCommands` 属性。活动命令通过在命令不使用时延迟它们的加载，来加快Atom的启动。所以把现有的命令移除，并在 `activationCommands` 中添加 `ascii-art:convert` ：

```
"activationCommands": {
  "atom-workspace": "ascii-art:convert"
}
```

首先，通过命令面板中的"Window: Reload"命令重新加载窗口，现在你可以执行"Ascii Art: Convert"命令了，它会输出"Hello, World!"。

添加快捷键

现在我们来添加用于触发"ascii-art:convert"命令的快捷键。打开 `keymaps/ascii-art.cson` ，添加一个键绑定来将 `ctrl-alt-a` 链接到 `ascii-art:convert` 命令上。由于你不需要预设的键绑定，你可以删除它们。

完成之后它应该像这样：

```
'atom-text-editor':
  'ctrl-alt-a': 'ascii-art:convert'
```

现在重新加载窗口，并验证快捷键是否工作。

添加字符画

现在我们需要将被选字符转换为字符画。为了完成它我们使用npm中的 `figlet` node模块。打开 `package.json` ，添加figlet的最新版本到 `dependencies` 中：

```
"dependencies": {
  "figlet": "1.0.8"
}
```

保存文件之后从命令面板运行“Update Package Dependencies: Update”。这会自动安装包的node模块依赖，在这个例子中只有figlet。无论什么时候你更新了 `package.json` 文件中的 `dependencies` 字段，你都要需要运行“Update Package Dependencies: Update”命令。

如果由于某种原因没有生效，你会看到“Failed to update package dependencies”这样的消息，并且会找到一个你的目录下有个新的 `npm-debug.log` 文件。这个文件会告诉你具体哪里有错误。

现在在 `lib/ascii-art.coffee` 中请求（require） `figlet` node模块，并且将被选文本转换成字符画来代替插入“Hello, World!”。

```
convert: ->
  if editor = atom.workspace.getActiveTextEditor()
    selection = editor.getSelectedText()

    figlet = require 'figlet'
    font = "o8"
    figlet selection, {font: font}, (error, art) ->
      if error
        console.error(error)
      else
        editor.insertText("\n#{art}\n")
```

重新加载编辑器，选择编辑器窗口中的一些文本，并按下 `ctrl-alt-a` ，取而代之的是，它会被替换成一个滑稽的字符画版本。

在这个例子中，我们需要快速查看一些新的东西。首先是 `editor.getSelectedText()` ，像你猜的那样，返回当前选中的文本。

之后我们调用Figlet的代码，来将它转换成别的东西，并使用 `editor.insertText()` 用它替换当前选中的文本。

小结

在这一节中，我们编写了一个无UI的包，用于获取选中文本并替换为处理过的版本。它可能会对创建文本提示和检查工具有帮助。

创建主题

Atom的界面使用HTML渲染，并且通过Less来定义样式，它是CSS的超集。不要担心之前从未听说过Less，它类似于CSS，但是带有一些便捷的扩展。

Atom支持两种主题：UI和语法。UI主题为树视图、选择夹、下拉列表和状态栏之类的元素定义样式。语法主题为编辑器中的代码定义样式。

主题可以从设置视图安装和修改，你可以选择 `Atom > Preferences…` 菜单，然后在左侧的侧栏中选择“Install”和“Theme”部分来打开它。

开始

主题是十分直截了当的，但是如果你在开始之前熟悉一些事情，会很有用处：

- Less是CSS的超集，但是它拥有一些像变量这样便利的特性。如果你并不熟悉它的语法，花几分钟在这里熟悉它。
- 你也可能想要复习一遍package.json的概念。这个文件帮助你把主题分布给其它用户。
- 你的主题中的“package.json”包必须含有一个 `theme` 键，值为 `ui` 或者 `syntax` ，为了让Atom识别为主题。
- 你可以在atom.io上面找到现有的主题，或者建立它们的分支（fork）。

创建语法主题

让我们来创建你的第一个主题。

按下 `cmd-shift-P` 来开始，并且输入“Generate Syntax Theme”就会一个新的主题包。选择“Generate Syntax Theme”之后，Atom会询问你要把主题新建在哪个目录下。我们把要创建的主题叫做“motif-syntax”。提示：语法主题应该以“-syntax”结尾。

然后Atom会弹出一个窗口展示motif-syntax主题，带有一些预先创建的文件和文件夹。如果你打开设置视图（ `cmd-,` ），并且访问左边的Themes部分，你会看到Syntax Theme下拉列表中列出了Motif。从菜单中选择它来加载。现在打开新的编辑器之后，你应该看到motif-syntax主题被激活了。

打开styles/colors.less来修改预先定义的各个颜色变量。例如，把 `@red` 变成 `#f4c2c1` 。

接着打开styles/base.less来修改预先定义的选择器。选择器为编辑器中不同部分定义样式，例如注释、字符串和侧栏中的行号。

例如，可以把 `.gutter``background-color` 设置为 `@red` 。

通过按下`cmd-alt-ctrl-l`重启Atom，来在Atom窗口中查看你的修改。这真是极好的。

提示：你可以通过在dev模式中打开新窗口，来避免查看你所做的修改时重启Atom。来命令行中运行`atom --dev .`来打开dev模式的Atom窗口。也可以按下`cmd-shift-o`或者打开`View > Developer > Open in Dev Mode`菜单来执行。当你编辑你的主题时，修改会立即表现出来。

建议不要在你的语法主题中指定`font-family`，因为会覆盖Atom设置中的`Font Family`字段。如果你仍旧想要推荐一款适合你主题的字体，我们推荐你在README文件中这么做。

创建界面主题

界面主题必须提供`ui-variables.less`文件，它包含了核心主题提供的所有变量。这些在"主题变量"一节会提到。

执行以下步骤来创建UI主题：

1. 创建以下仓库之一的分支：
    - atom-dark-ui
    - atom-light-ui
2. 克隆分支到本地文件系统
3. 在主题的目录中打开命令行
4. 在命令行中通过运行`atom --dev .`，或者点击`View > Developer > Open in Dev Mode`菜单，以dev模式打开你的新主题
5. 在主题的`package.json`文件中修改主题的名字
6. 以`-ui`结尾的名字命名你的主题，例如`super-white-ui`
7. 运行`apm link`来把你的主题符号链接到`~/.atom/packages`
8. 使用`cmd-alt-ctrl-L`重启Atom
9. 通过设置视图的Themes部分中的UI Theme下拉列表来开启主题
10. 做一些修改。由于你在dev模式窗口下打开主题，修改会立即在编辑器中反映，并不需要重启。

开发的工作流

下面是一些使主题开发更快速更简单的工具。

即时重启

在你修改你的主题之后，按下`cmd-alt-ctrl-L`来重启不是十分理想。在dev模式的Atom窗口下，Atom支持样式的即时更新。

要想开启dev模式的窗口：

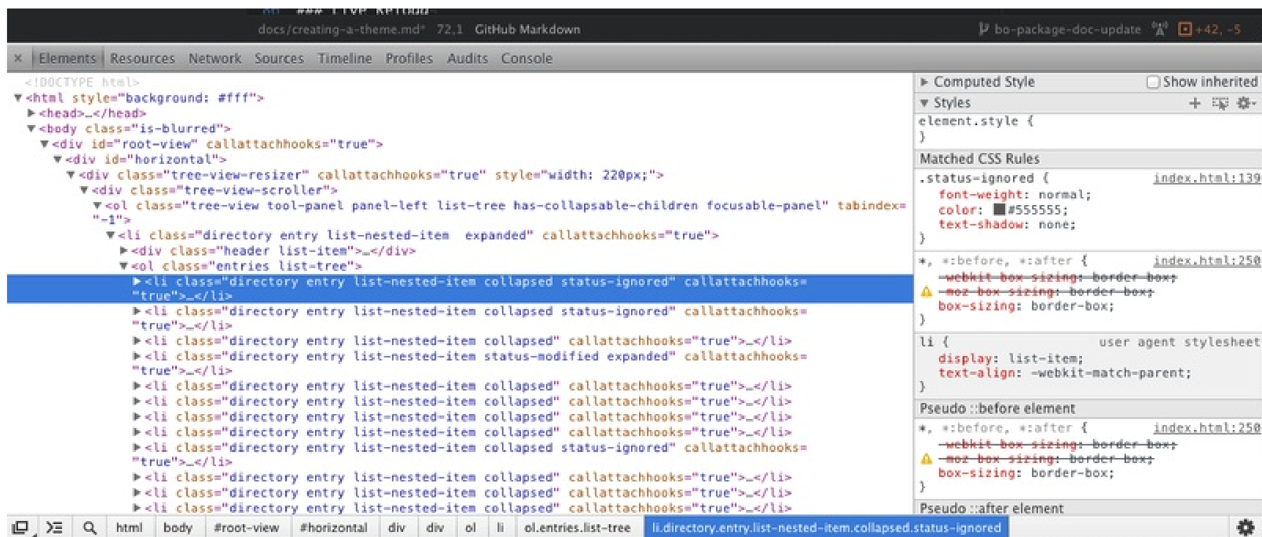
1. 通过选择 `View > Developer > Open in Dev Mode` 菜单，或者按下 `cmd-shift-o` 快捷键来直接在dev模式窗口中打开你的主题。
2. 修改你的主题并保存它。你的修改应该会马上应用。

如果你想要在任何时候都重新加载全部的样式，你可以使用 `cmd-ctrl-shift-r` 快捷键。

开发者工具

Atom基于Chrome浏览器，并且支持Chrome开发者工具。你可以选择 `View > Toggle Developer Tools` 菜单，或者使用 `cmd-alt-i` 快捷键来打开它。

开发者工具允许你查看各个元素，以及他们的CSS属性。

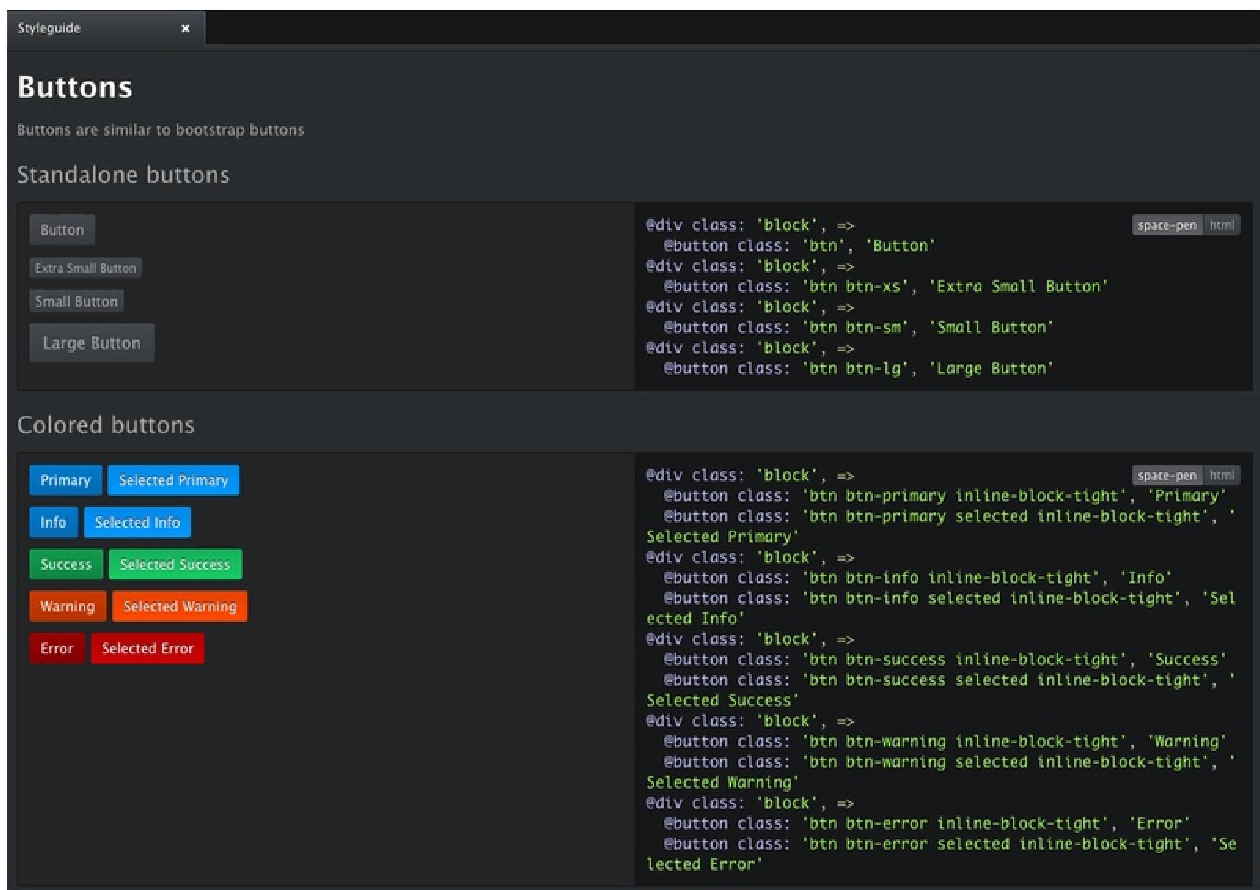


简单介绍请查看Google的扩展教程。

Atom 样式指南

如果你在创建一个界面主题，你可能想要一种方式来查看你的主题如何影响系统中的组件。样式指南是一个页面，里面渲染了所有Atom支持的组件。

打开命令面板（ `cmd-shift-P` ）寻找“styleguide”，或者使用 `cmd-ctrl-shift-g` 快捷键来打开样式指南。



主题变量

Atom的UI提供了一些变量，你可以在你自己的主题或者包中使用它们。

在主题中使用

每个自定义的主题都要指定 `ui-variables.less` 文件，其中定义了所有下面的变量。主题列表中最上面的主题会被加载，以及可供导入。

在包中使用

在任何你的包的 `.less` 文件中，你可以通过从Atom导入 `ui-variables` 文件来访问主题变量。

你的包应该只指定结构化的样式，并且它们应该全部来自样式指南。你的包不应该指定颜色、内边距（padding）、或者使用绝对像素的任何东西。你应该使用主题变量来代替它。如果你遵循了这一点，你的包将会在任何主题下都表现得很好。

这里是一个 `.less` 文件的例子，一个包可以使用以下主题变量来定义：

```
@import "ui-variables";

.my-selector {
  background-color: @base-background-color;
  padding: @component-padding;
}
```

变量

文本颜色

- `@text-color`
- `@text-color-subtle`
- `@text-color-highlight`
- `@text-color-selected`
- `@text-color-info` - 蓝色
- `@text-color-success` - 绿色
- `@text-color-warning` - 橙色或者黄色
- `@text-color-error` - 红色

背景颜色

- `@background-color-info` - 蓝色
- `@background-color-success` - 绿色
- `@background-color-warning` - 橙色或者黄色
- `@background-color-error` - 红色
- `@background-color-highlight`
- `@background-color-selected`
- `@app-background-color` - 所有编辑器组件下面的应用背景

组件颜色

- `@base-background-color` -
- `@base-border-color` -
- `@pane-item-background-color` -
- `@pane-item-border-color` -
- `@input-background-color` -
- `@input-border-color` -
- `@tool-panel-background-color` -
- `@tool-panel-border-color` -
- `@inset-panel-background-color` -
- `@inset-panel-border-color` -
- `@panel-heading-background-color` -
- `@panel-heading-border-color` -

- `@overlay-background-color` -
- `@overlay-border-color` -
- `@button-background-color` -
- `@button-background-color-hover` -
- `@button-background-color-selected` -
- `@button-border-color` -
- `@tab-bar-background-color` -
- `@tab-bar-border-color` -
- `@tab-background-color` -
- `@tab-background-color-active` -
- `@tab-border-color` -
- `@tree-view-background-color` -
- `@tree-view-border-color` -
- `@ui-site-color-1` -
- `@ui-site-color-2` -
- `@ui-site-color-3` -
- `@ui-site-color-4` -
- `@ui-site-color-5` -

组件尺寸

- `@disclosure-arrow-size` -
- `@component-padding` -
- `@component-icon-padding` -
- `@component-icon-size` -
- `@component-line-height` -
- `@component-border-radius` -
- `@tab-height` -

字体

- `@font-size` -
- `@font-family` -

图标

Atom自带了Octicons的图标集。使用它们来添加图标到你的包中。

使用方法

Octicons在Atom中的使用方法不同于标准用法。最大的不同是图标class的名字。你应该使用更加通用的 `icon icon-` 前缀，而不是 `octicon octicon-` 前缀。

例如，要想添加monitor图标，在你的标记中使用 `icon icon-device-desktop` class：

```
<span class="icon icon-device-desktop"></span>
```

或者你可以像这样使用SpacePen：

```
@span class: 'icon icon-device-desktop'
```

尺寸

Octicons在 `16px` 的 `font-size` 下最美观。通常条件下就是这样使用的，所以你不用担心。如果你更喜欢不同的图标尺寸，尝试使用16的倍数（比如 `32px` 或者 `48px` ）来显示得更清晰。在此之间的尺寸也可以，但可能对于一些带直线的图标会显示得很模糊。

易用性

虽然图标会让你的UI更具有视觉感，不和文本标签一起使用的时候，就很难猜出它的意思。对于空间不足以放下文本标签的情况，考虑放置一个鼠标覆盖时显示的提示框。或者一个更巧妙的 `title="label"` 属性也会有所帮助。

调试

Atom拱了一些工具来帮助你理解预料之外的行为和调试问题。这篇指南介绍了一些工具和方法用于帮助你调试，以及提供了一些提交工单（issue）时的帮助信息。

升级到最新版本

你可能遇到了在最新版本已经修复的问题。

如果你从源码中编译Atom，需要拉回（pull）master的最新版本并重新构建。

如果你使用发布版本，检查你所使用的Atom是什么版本：

```
$ atom --version
0.178.0-37a85bc
```

访问发布列表来查看是否有更新的版本。你可以从发布页面下载Atom来升级到最新的版本，或者通过应用中的自动升级工具。应用中的自动升级工具在你重启Atom，或者点击 `Atom > Check for Update` 菜单之后，会检查和下载新的版本。

检查链接的包

如果你开发或者发布Atom的包，可能会有一些遗留的包链接到 `~/.atom/packages` 或者 `~/.atom/dev/packages` 目录下。你可以使用：

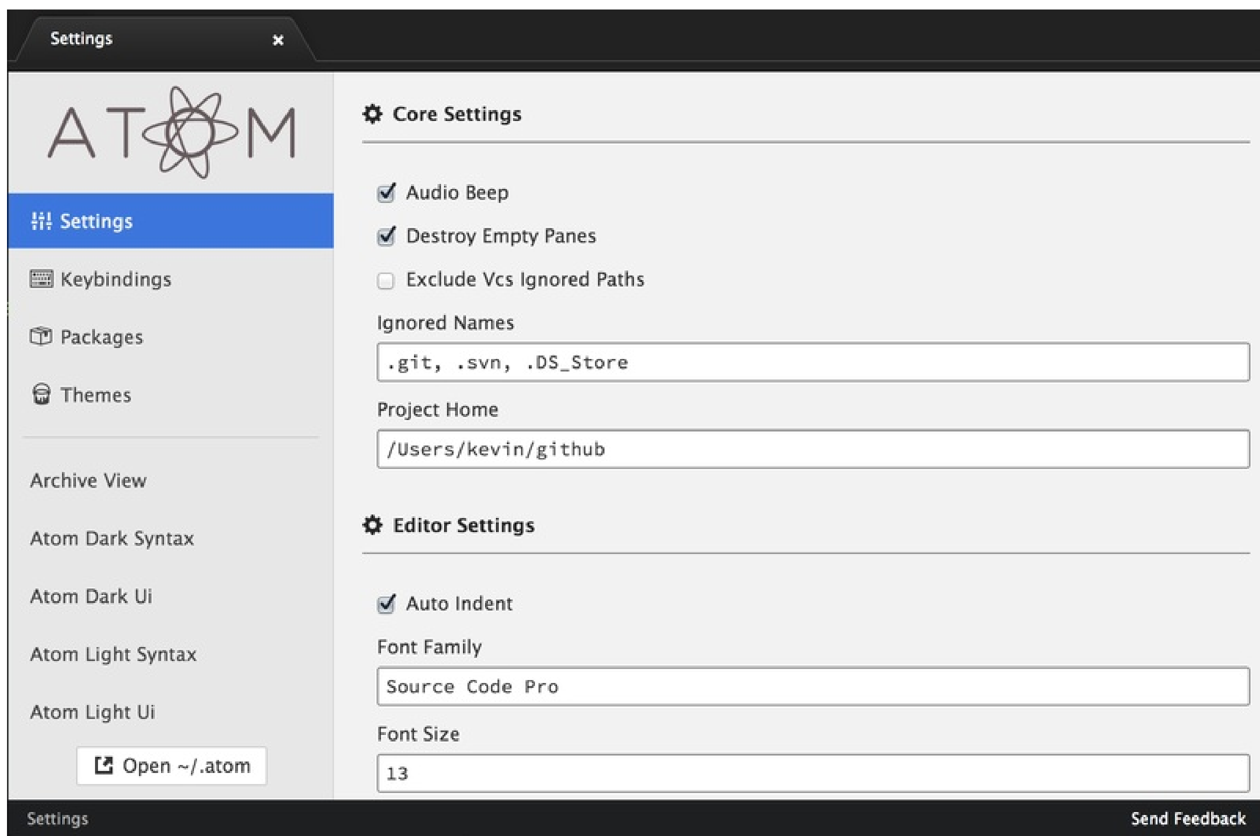
```
$ apm links
```

来列出所有链接的开发包。你可以使用 `apm unlink` 命令来移除链接，详见 `apm unlink --help` 。

检查Atom和包的设置

在一些情况下，预料之外的行为可能是Atom或者某个包中的错误配置或者缺少配置造成的。

使用 `cmd-,` 来打开Atom的设置视图，或者 `Atom > Preferences` 来打开菜单选项。

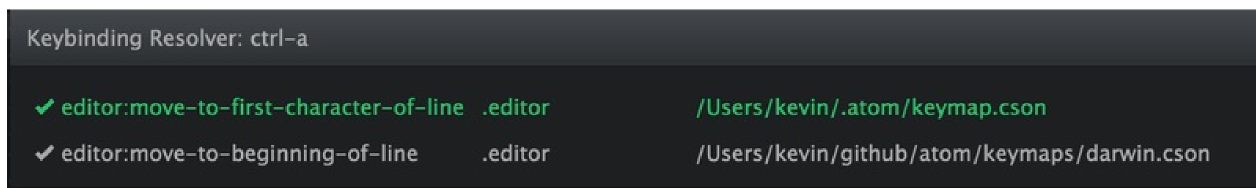


在设置面板中检查Atom的设置，每个选项在这里都会有个详细的描述。例如，如果你希望Atom使用硬tab（真的tab）而不是软tab（空格），你应该取消“Soft Tabs”选项。

由于Atom自带一些包并且你可以自己安装附加的包，检查所有包的列表以及他们的设置。例如，如果你喜欢移除编辑器中间的竖直线，禁用Wrap Guide包。另外，如果你希望Atom移除行尾的空白字符，或者确保文件末尾有个空行，你可以在Whitespace包的选项中设置。

检查快捷键

如果你按下快捷键之后没有执行命令，或者执行了错误的命令，那个键位的快捷键可能出了些问题。Atom自带keybinding resolver，一个小巧的包来帮助你理解执行了哪个快捷键。



keybinding resolver会向你展示现有快捷键的列表，列表中包含以下内容：

- 快捷键的命令
- 快捷键有效时，用于定义上下文的CSS选择器
- 快捷键被定义的文件

如果匹配到多个快捷键，Atom会根据选择器的特性和以及他们被加载的顺序来决定执行哪个快捷键。如果你想要触发的命令在keybinding resolver中列出，但是并没有执行，一般由以下两种原因造成：

- 快捷键并没有在选择器定义的上下文中使用。例如，你不能在树视图没有焦点的情况下触发“Tree View: Add File”命令。
- 有另一个快捷键具有更高的优先级。这通常发生在你安装的包的快捷键和现有的快捷键冲突的时候。如果这个包的快捷键具有更高的特异性的选择器，或者更晚被加载，它就会覆盖现有的快捷键。

Atom首先会加载核心功能的快捷键，之后才是用户定义的快捷键。由于用户定义的快捷键在随后加载，你可以使用 `keymap.cson` 文件来调整快捷键并解决问题。例如，你可以使用 `unset!` 指令来移除快捷键。

如果你发现一个包的快捷键优先级要高于核心功能包，向这个包的GitHub 仓库提交一个工单可能是个好主意。

查看是否在安全模式下出现问题

Atom绝大多数的功能都来源于你安装的包。一些情况下，这些包可能会导致预料之外的行为、问题或者性能问题。

从命令行在安全模式下启动Atom，来检查是否是你安装的一个包导致了问题：

```
$ atom --safe
```

这会启动Atom，但是并不会加载 `~/.atom/packages` 和 `~/.atom/dev/packages` 中的包。如果安全模式下问题不再出现，那么可能那些包之一导致了问题。

要弄清楚到底是哪个包导致了问题，正常启动Atom，并且按下 `cmd-,` 来打开设置。由于设置可以让你禁用每个包，你可以依次禁用每个包，直到问题不再发生。在你禁用每个包来确保问题不再出现之后，重启（ `cmd-q` ）或者重新加载（ `cmd-ctrl-alt-l` ）Atom。

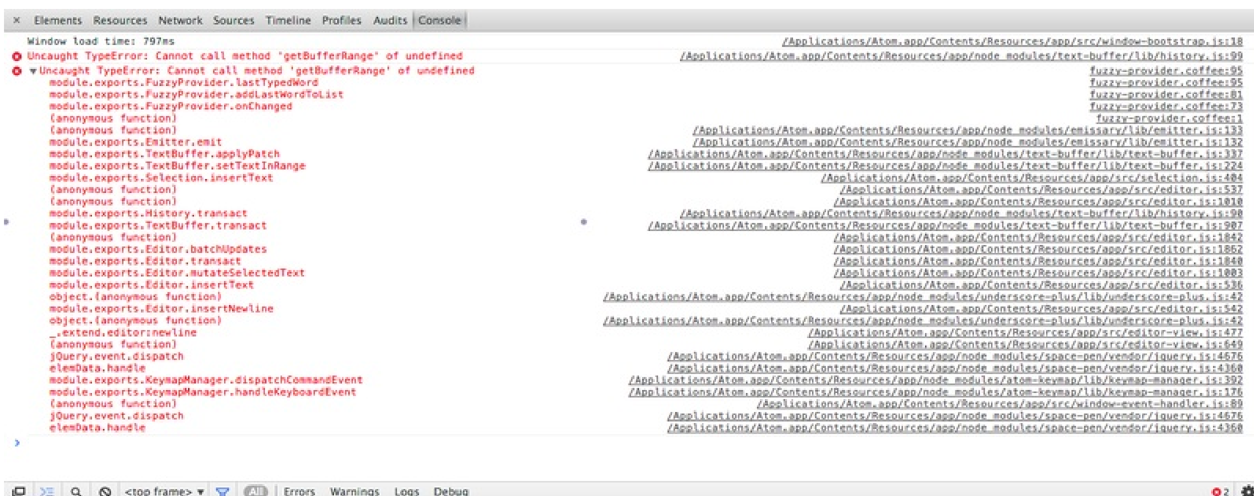
当你找到导致问题的包之后，你可以禁用或者卸载这个包，并且考虑向这个包的GitHub仓库提交工单。

检查你的配置文件

你可能在Atom的初始化脚本或者样式表中定义了一些个性化的功能。在一些情况中，这些个性化的调整可能会导致问题，所以清除这些文件，并重启Atom。

检查在开发者工具中的错误

当Atom中抛出了一个错误，开发者工具会在控制台标签页自动展示记录的错误。而如果开发者工具在错误触发之前打开，错误的整个栈轨迹会被记录：

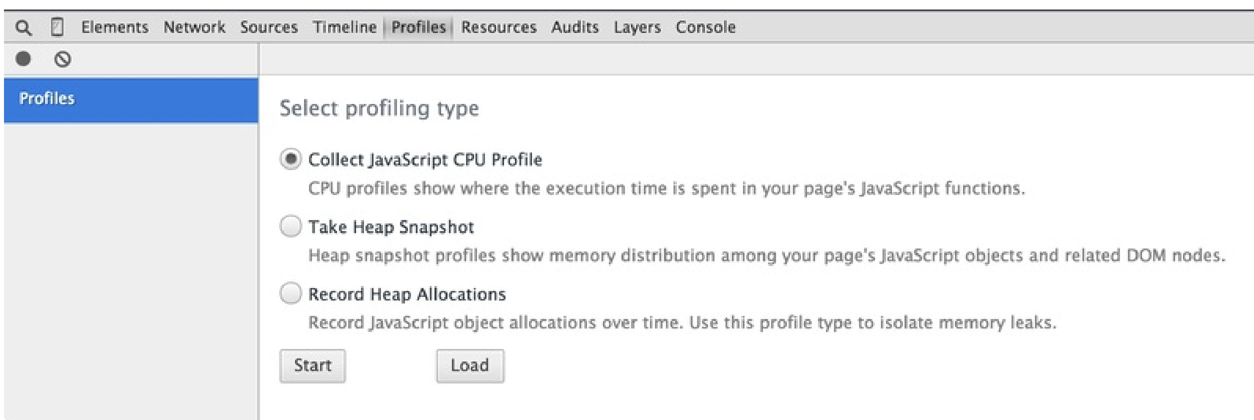


如果你可以重现这个错误，使用这种方法来得到全部的栈轨迹。栈轨迹可能会指向你的初始化脚本，或者安装的某个特定的包，可以禁用它们并且向Github仓库提交工单。

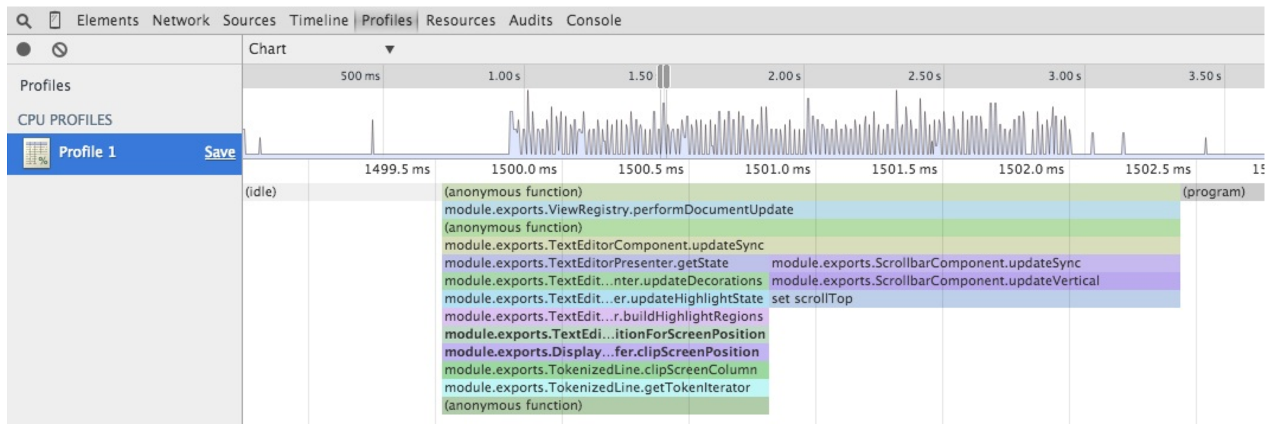
在开发者页面的CPU profiler中判断性能问题

如果你在特定的情况中发现了性能问题，如果报告中包含了Chrome的CPU profiler截图，提供了一些什么东西比较慢的洞察，你的报告会很有用处。

要运行profiler，在命令面板中打开开发者工具（“Window: Toggle Dev Tools”），访问 `Profiles` 标签页，选择 `Collect JavaScript CPU Profile` 并点击start按钮。



然后返回Atom并且执行速度慢的操作来做记录。结束之后按下stop按钮。切换到 `Chart` 视图，会出现一副记录操作的图片。尝试放大缓慢的部分，并且截图来包含到你的报告中。你也可以保存并发送profile数据，通过按下左边面板中的名字（例如 `Profile 1` ）旁边的 `Save` 按钮。



详见Chrome的CPU profiling文档。

检查你是否安装了开发工具链

如果你在使用 `apm install` 安装一个包时出现问题，可能是因为那个包依赖了使用本地代码的库。所以你需要安装C++编译器和Python来安装它。

你可以运行 `apm install --check` 来查看Atom是否能够在你的机器上编译本地代码。

关于更多信息，在构建指导中查看你的平台上需要先满足的条件。

编写 spec

我们已经通过一些例子查看并编写了一些spec，现在是更进一步查看spec框架本身的时候了。确切地说，你在Atom中如何编写测试呢？

Atom使用Jasmine作为spec框架。任何新的功能都要拥有specs来防止回归。

创建新的 spec

Atom的spec和包的spec都要添加到它们各自的 `spec` 目录中。下面的例子为Atom核心创建了一个spec。

创建spec文件

spec文件必须以 `-spec` 结尾，所以把 `sample-spec.coffee` 添加到 `atom/spec` 中。

添加一个或多个 `describe` 方法

`describe` 方法有两个参数，一个描述和一个函数。以 `when` 开始的描述通常会解释一个行为；而以方法名称开头的描述更像一个单元测试。

```
describe "when a test is written", ->
  # contents
```

或者

```
describe "Editor::moveUp", ->
  # contents
```

添加一个或多个 `it` 方法

`it` 方法也有两个参数，一个描述和一个函数。尝试去让 `it` 方法长于描述。例如， `this should work` 的描述并不如 `it this should work` 便于阅读。但是 `should work` 的描述要好于 `it should work` 。

```
describe "when a test is written", ->
  it "has some expectations that should pass", ->
    # Expectations
```

添加一个或多个预期

了解预期（expectation）的最好方法是阅读Jasmine的文档。下面是个简单的例子。

```
describe "when a test is written", ->
  it "has some expectations that should pass", ->
    expect("apples").toEqual("apples")
    expect("oranges").not.toEqual("apples")
```

异步的spec

编写异步的spec刚开始会需要些技巧。下面是一些例子。

Promise

在Atom中处理Promise更加简单。你可以使用我们的 `waitsForPromise` 函数。

```
  describe "when we open a file", ->
    it "should be opened in an editor", ->
      waitsForPromise ->
        atom.workspace.open('c.coffee').then (editor) ->
          expect(editor.getPath()).toContain 'c.coffee'
```

这个方法可以在 `describe` 、 `it` 、 `beforeEach` 和 `afterEach` 中使用。

```
describe "when we open a file", ->
  beforeEach ->
    waitsForPromise ->
      atom.workspace.open 'c.coffee'

  it "should be opened in an editor", ->
    expect(atom.workspace.getActiveTextEditor().getPath()).toContain 'c.coffee'
```

如果你需要等待多个promise，对每个promise使用一个新的 `waitsForPromise` 函数。（注意：如果不用 `beforeEach` 这个例子会失败）

```
describe "waiting for the packages to load", ->

  beforeEach ->
    waitsForPromise ->
      atom.workspace.open('sample.js')
    waitsForPromise ->
      atom.packages.activatePackage('tabs')
    waitsForPromise ->
      atom.packages.activatePackage('tree-view')

  it 'should have waited long enough', ->
    expect(atom.packages.isPackageActive('tabs')).toBe true
    expect(atom.packages.isPackageActive('tree-view')).toBe true
```

带有回调的异步函数

异步函数的Spec可以 `waitsFor` 和 `runs` 函数来完成。例如：

```
describe "fs.readdir(path, cb)", ->
  it "is async", ->
    spy = jasmine.createSpy('fs.readdirSpy')

    fs.readdir('/tmp/example', spy)
    waitsFor ->
      spy.callCount > 0
    runs ->
      exp = [null, ['example.coffee']]
      expect(spy.mostRecentCall.args).toEqual exp
      expect(spy).toHaveBeenCalledWith(null, ['example.coffee'])
```

访问Jasmine文档)来了解更多关于异步测试的细节。

运行 spec

大多数情况你会想要通过触发 `window:run-package-specs` 来运行spec。这个命令不仅仅运行包的spec，还运行了Atom的核心spec。它会运行当前项目spec目录中的所有spec。如果你想要运行Atom的核心spec和所有默认包的spec，触发 `window:run-all-specs` 命令。

要想运行spec的一个有限的子集，使用 `fdescribe` 和 `fit` 方法。你可以使用它们来聚焦于单个或者几个spec。在上面的例子中，像这样聚焦于一个独立的spec：

```
describe "when a test is written", ->
  fit "has some expectations that should pass", ->
    expect("apples").toEqual("apples")
    expect("oranges").not.toEqual("apples")
```

在CI中运行

在CI环境，类似Travis和AppVeyor中运行spec现在非常容易。详见文章“Travis CI For Your Packages”和“AppVeyor CI For Your Packages”。

从Textmate中转换

可能在Textmate中有你喜欢或者使用过的主题和语法，并且你想要把它们转换到Atom中。如果是这样的话，你很幸运，因为有很多工具可以用来转换它们。

转换 TextMate Bundle

TextMate bundle的转换允许你在Atom中使用TextMate的偏好、代码段和配色。

让我们来为R语言转换TextMate bundle。你可以在Github上面找到其它现存的TextMate bundle。

你可以使用以下命令来转换R bundle：

```
$ apm init --package ~/.atom/packages/language-r \
  --convert https://github.com/textmate/r.tmbundle
```

现在你可以浏览 `~/.atom/packages/language-r` 来查看转换后的bundle。

新的包已经可以使用了，运行Atom并在编辑器中打开一个 `.r` 文件，就可以看到效果。

转换TextMate 主题

这一节会介绍如何把TextMate主题转换成Atom主题。

差异

TextMate主题使用plist文件，而Atom使用CSS或者Less来定义编辑器中语法和UI的样式。

转换主题的工具首先解析主题的plist文件，然后创建与之对应的CSS规则和属性，它们为Atom定义相似的样式。

转换主题

下载你想要转换的主题，你可以在Textmate的网站浏览已有的Textmate主题。

现在假设你已经将主题下载到 `~/Downloads/MyTheme.tmTheme` ，你可以使用以下命令来转换主题：

```
$ apm init --theme ~/.atom/packages/my-theme \
  --convert ~/Downloads/MyTheme.tmTheme
```

之后你可以浏览`~/.atom/packages/my-theme`来查看转换后的主题。

启用主题

你的主题安装到`~/.atom/packages`之后，你可以通过运行Atom并且选择`Atom > Preferences…`菜单来开启它。

选择左侧边栏上的`Themes`链接，并且选择`My Theme from the Syntax Theme`下拉菜单来开启你的新主题。

你的主题现在被启用了，可以打开编辑器来查看效果。

在Atom背后

在我们编写了一些主题和包之后，让我们花一些时间来深入了解一些Atom的工作方式。在这一章中，我们会更进一步了解Atom中独特的内部API和系统，甚至查看一些源码来看一看它们是如何很好地工作的。

配置API

读取配置

如果你想要编写一个可配置的包，你需要通过 `atom.config` 来整体读取配置，或者通过 `atom.config.get` 来读取一个具有命名空间的配置键的当前值。

```
# read a value with `config.get`
@showInvisibles() if atom.config.get "editor.showInvisibles"
```

或者通过 `atom.config.observe` 来跟踪任何视图对象产生的修改。

```
{View} = require 'space-pen'

class MyView extends View
  attached: ->
    @fontSizeObserveSubscription =
      atom.config.observe 'editor.fontSize', (newValue, {previous}) =>
        @adjustFontSize()

  detached: ->
    @fontSizeObserveSubscription.dispose()
```

`atom.config.observe` 方法会使用特定键路径的当前值立即调用提供的回调函数，并且以后当这个值发生改变时也会调用。如果你只希望在下次这个值改变的时候调用回调，使用 `atom.config.onDidChange` 来代替它。

订阅（Subscription）方法会返回一个一次性的订阅对象。注意上面的例子中，我们如何将订阅保存在 `@fontSizeObserveSubscription` 实例变量中，以及视图被分离的时候如何处理它。你可以添加多个订阅到 `[CompositeDisposable](https://atom.io/docs/api/latest/CompositeDisposable)` 中来将它们组合到一起。当视图被分离的时候你可以把它处理掉。

写入配置

虽然 `atom.config` 的数据在启动时才从 `~/.atom/config.cson` 加载，但你可以通过 `atom.config.set` 用编程的方式对其写入。

```
# basic key update
atom.config.set("core.showInvisibles", true)
```

如果你通过特定键路径来访问包的配置，你也可能像将它们和你包里面的主模块的schema关联起来。更多schema的细节请阅读配置API文档。

深入键表（keymap）

键表文件是以JSON或者CSON编码的文件，其中含有嵌套的哈希表。它们的工作方式像是样式表，但是它们指定匹配选择器的元素的快捷键的作用，而不是应用样式属性。下面是一些快捷键的例子，它们在 `atom-text-editor` 元素上按下时生效：

```
'atom-text-editor':
  'cmd-delete': 'editor:delete-to-beginning-of-line'
  'alt-backspace': 'editor:delete-to-beginning-of-word'
  'ctrl-A': 'editor:select-to-first-character-of-line'
  'ctrl-shift-e': 'editor:select-to-end-of-line'
  'cmd-left': 'editor:move-to-first-character-of-line'

'atom-text-editor:not([mini])':
  'cmd-alt-[': 'editor:fold-current-row'
  'cmd-alt-]': 'editor:unfold-current-row'
```

在第一个选择器底下绑定了一些快捷键，将特定的键位通配符映射到命令上面。当一个 `atom-text-editor` 元素获得焦点，并且 `cmd-delete` 被按下，一个叫做 `editor:delete-to-beginning-of-line` 的自定义DOM事件会在 `atom-text-editor` 元素上面触发。

第二个选择器分组也指向了编辑器，但是只是没有 `mini` 属性的编辑器。在这个例子中，代码折叠的命令在迷你编辑器中毫无意义，所以选择器将它们限制于普通的编辑器中。

键位通配符（keystroke pattern）

键位通配符表示一个或者多个键位，带有可选的辅助键（modifier key）。例如 `ctrl-w v` 和 `cmd-shift-up` 。键位由下面的符号组成，以 `-` 分隔。一个多种键位的通配符可以表示为以空格分割的键位通配符。

类型	例子
字符的字面值	`a` `4` `$`
辅助键	`cmd` `ctrl` `alt` `shift`
特殊键	`enter` `escape` `backspace` `delete` `tab` `home` `end` `pageup` `pagedown` `left` `right` `up` `down`

命令

命令是自定义的DOM事件，当一个键位匹配到绑定的快捷键时触发。这可以让UI代码来监听具名的命令，而不需要指定触发它的特定的快捷键。例如，下面的代码创建了一个命令来向编辑器插入当前日期：

```
atom.commands.add 'atom-text-editor',
  'user:insert-date': (event) ->
    editor = @getModel()
    editor.insertText(new Date().toLocaleString())
```

`atom.commands` 指向全局 `{CommandRegistry}` 的实例，所有命令在它里面设置，并且可以通过命令面板来获取。

当你想要绑定新的快捷键时，使用命令面板（ `ctrl-shift-p` ）来看一看在一个具有焦点的上下文中，什么命令正在被监听，是十分有用的。遵循一个简单的算法使得命令会很“人性化”，所以 `editor:fold-current-row` 命令会显示为“Editor: Fold Current Row”。

“组合”命令

一个很常见的问题是，“我如何使用一个快捷键来执行两个或者更多命令？”Atom并不直接支持这一需求，但是我们可以通过创建一个自定义命令，它执行你想要的多个操作，并且为这个命令创建一个快捷键来解决。例如，假设我想创建一个“组合”命令，选取并剪切一行。你可以在 `init.coffee` 中添加一下代码：

```
atom.commands.add 'atom-text-editor', 'custom:cut-line', ->
  editor = atom.workspace.getActiveTextEditor()
  editor.selectLinesContainingCursors()
  editor.cutSelectedText()
```

然后我们想要把这个命令关联到 `alt-ctrl-z` 上去，你应该添加以下内容到键表中：

```
'atom-text-editor':
  'alt-ctrl-z': 'custom:cut-line'
```

特异性（优先级）和层级顺序

就像这个应用了CSS样式的例子，当很多快捷键的绑定匹配到一个元素的时候，冲突通过选择最特别的选择器来解决。如果两个匹配到的选择器具有相同的特异性，在层级中出现顺序靠后的选择器的快捷键会优先执行。

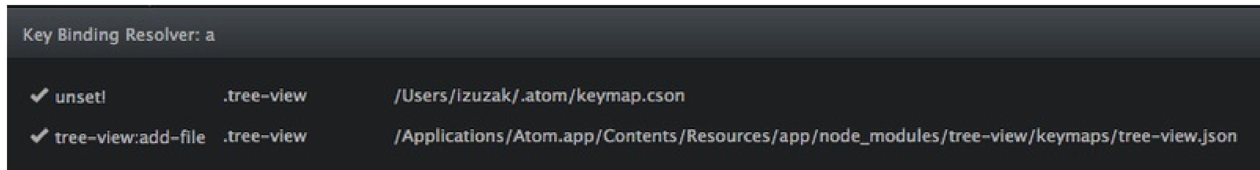
当前，没有任何方法在一个单独的键表中指定快捷键的顺序，因为JSON的对象是无序的。我们最终打算为键表引入一个自定义类似CSS的文件格式来允许在单个文件中排序。到目前为止，我们可以选择性解决一些情况，其中选择器的顺序由把键表分开放到两个文件中来严格规定。就像 `snippets-1.cson` 和 `snippets-2.cson` 。

移除快捷键

当键表系统遇到了以`unset!`作为快捷键的命令，它就会像没有绑定匹配到当前键位序列一样，继续从它的父节点中寻找。如果你想移除一个你不再用到的快捷键，例如Atom核心中的或者包中的快捷键，应该直接使用`unset!`。

例如，下面的代码移除了树视图上`a`的快捷键，它一般会触发`tree-view:add-file`命令：

```
'.tree-view':
  'a': 'unset!'
```



强制Chrome处理本地快捷键

如果你想要在一个提供的快捷键上强制执行本地浏览器的行为，直接使用`native!`作为绑定的命令。这会在启动本地输入元素的正确行为时比较有用。例如，如果你在一个元素上面应用了`.native-key-bindings` class，所有由浏览器处理的快捷键都会绑定为`native!`。

重载快捷键

一些情况下需要把多个动作依次放到同一个快捷键下面。一个例子就是代码段的包，代码段由输入一个类似`for`的前缀之后按下`tab`来插入。每次`tab`按下的时候，如果光标前面的文字存在对应的代码段，我们想要执行代码来展开代码段。如果代码段并不存在，我们希望`tab`插入空白字符。

要实现成这样，代码段的包利用了代表`snippets:expand`命令的事件对象的`.abortKeyBinding()`方法。

```
# pseudo-code
editor.command 'snippets:expand', (e) =>
  if @cursorFollowsValidPrefix()
    @expandSnippet()
  else
    e.abortKeyBinding()
```

当事件处理器观察到光标前面并没有一个有效的前缀时，会调用`e.abortKeyBinding()来告诉键表系统继续寻找另一个匹配到的绑定。

详细步骤：按键事件如何映射到命令

- 按键事件出现在获得焦点的元素上面。

- 由获取焦点的元素开始，键表会向上搜索，直到文档的根元素，寻找最具特异性的CSS选择器，它匹配当前DOM元素并且含有匹配按键事件的快捷键通配符。
- 找到匹配的快捷键通配符之后，搜索就结束了，并且与通配符绑定的命令会在当前元素上触发。
- 如果在触发的事件对象上调用了 `.abortKeyBinding()` ，会恢复搜索，在相同元素上触发下一个最具特异性的CSS选择器上绑定的事件，或者继续向上搜索。
- 如果找不到任何快捷键，事件通常就会由Chrome来处理。

作用域设置、作用域和作用域描述符

Atom支持语言特定的设置。你可以在Markdown文件中软换行，或者在Python中把tab的宽度设置为4。

语言特定的设置只是一些东西的子集，我们把它叫做“作用域设置”。作用域设置可以只作用于一类特定的语法符号。比如你可以仅仅对Ruby的注释，Markdown中的代码段，或者JavaScript函数名称进行设置。

语法符号中的作用域名称

编辑器的每个符号都有一系列的作用域名称。例如，前面提到的JavaScript函数可能拥有作用域 `function` 和 `name` 。一个左括号可能拥有 `punctuation` 、 `parameters` 和 `begin` 作用域。

作用域的名称就像CSS中的class一样工作。事实上，编辑器中的作用域名称作为CSS的class附加到符号的DOM节点。

比如这段JavaScript代码：

```
function functionName() {
  console.log('Log it out');
}
```

在开发工具中，第一行的标记就像这样：

```
▼<span class="source js">
  ▼<span class="meta function js">
     <span class="storage type function js">function</span>
     <span class="entity name function js">functionName</span>
     <span class="punctuation definition parameters begin js">(</span>
     <span class="punctuation definition parameters end js">)</span>
   </span>
   <span class="meta brace curly js">{</span>
 </span>
```

span标签上的所有class名称都是作用域名称。任何作用域名称都用于指向一个设置的值。

作用域选择器

作用域选择器允许你指向特性符号，就像CSS选择器指向DOM中特定的节点。这里是一些例子：

```
'.source.js' # selects all javascript tokens
'.source.js .function.name' # selects all javascript function names
'.function.name' # selects all function names in any language
```

`Config::set` 接受一个 `scopeSelector` 。如果你想要对JavaScript 函数名称进行设置，你可以向它提供一个js函数名称的 `scopeSelector` ：

```
atom.config.set('.source.js .function.name', 'my-package.my-setting', 'special value')
```

作用域描述符

作用域描述符是一个对象，它封装了一个字符串数组。数组描述了从语法树根节点到符号的路径，包含整个路径的所有作用域名称。

在上面的JavaScript例子中，函数名称符号的作用域描述符应该为：

```
['source.js', 'meta.function.js', 'entity.name.function.js']
```

`Config::get` 接受一个 `scopeDescriptor` ，你可以从作用在JavaScript函数名称的设置中获取值，通过：

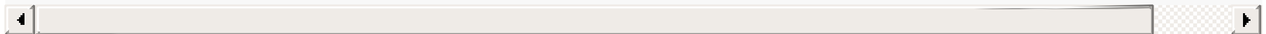
```
scopeDescriptor = ['source.js', 'meta.function.js', 'entity.name.function.js']
value = atom.config.get(scopeDescriptor, 'my-package.my-setting')
```

但是你并不需要手动生成作用域描述符。有一些可用的方法来从编辑器获取作用域描述符：

- `Editor::getRootScopeDescriptor` 获取语言的描述符，例如 `[".source.js"]` 。
- `Editor::scopeDescriptorForBufferPosition` 获取缓冲区中特定位置的描述符。
- `Cursor::getScopeDescriptor` 获取光标处的描述符。例如，如果光标在例子中的方法名称上面，会返回 `["source.js", "meta.function.js", "entity.name.function.js"]` 。

让我们使用这些方法来回顾我们的例子：

```
editor = atom.workspace.getActiveTextEditor()
cursor = editor.getLastCursor()
valueAtCursor = atom.config.get(cursor.getScopeDescriptor(), 'my-package.my-setting')
valueForLanguage = atom.config.get(editor.getRootScopeDescriptor(), 'my-package.my-settin
```



Atom中的序列化

当一个窗口被刷新，或者从上一次会话恢复的时候，视图和它相关的对象会从JSON表达式中反序列化，它们在窗口上一次关闭时储存。要使你自己的视图和对象兼容刷新，你需要让它们很好地执行序列化和反序列化。

包的序列化钩子

你的包的主模块可以选择包含一个`serialize`方法，它在你的包反激活之前被调用。你应该返回JSON，它会交还给你作为下次调用`activate`的参数。在下面的例子中，这个包将一个`MyObject`的实例在刷新过程中保持相同的状态。

```
module.exports =
  activate: (state) ->
    @myObject =
      if state
        atom.deserializers.deserialize(state)
      else
        new MyObject("Hello")

  serialize: ->
    @myObject.serialize()
```

序列化方法

```
class MyObject
  atom.deserializers.add(this)

  @deserialize: ({data}) -> new MyObject(data)
  constructor: (@data) ->
  serialize: -> { deserializer: 'MyObject', data: @data }
```

.serialize()

你想要序列化的对象需要实现`.serialize()`，这个方法需要返回一个序列化的对象。而且它必须包含一个叫做`deserializer`的键，它的值为一个已注册的反序列化器的名字，它能够转换对象剩余的数据。它通常是类的名称本身。

@deserialize(data)

另一方面是`deserialize`方法，它通常是个类级的方法，位于实现`serialize`方法相同的类中。它的作用是将一个上一次`serialize`调用返回的状态对象转化为一个真正的对象。

atom.deserializers.add(class)

你需要在你的类中调用`atom.deserializers.add`方法来使它对反序列化系统可见。现在你可以带着`serialize`返回的状态调用全局的`deserialize`方法，你的类中的`deserialize`会自动被选择。

版本控制

```
class MyObject
  atom.deserializers.add(this)

  @version: 2
  @deserialize: (state) -> ...
  serialize: -> { version: @constructor.version, ... }
```

你的可序列化类可以带有一个可选的`@version`类级属性，并且在序列化的状态中持有`version`键。反序列化的时候，Atom只在版本匹配的时候尝试对其反序列化，否则返回`undefined`。我们计划在未来实现一个迁移系统，但是现在至少能防止你对旧的状态反序列化。

开发Node模块

Atom中的一些包是Node模块，而不是Atom的包。如果你想要修改这些Node模块，例如 `atom-keymap` ，你需要把它们链接到不同于普通Atom包的开发环境中。

把Node模块链接到你的Aton开发环境

下面是运行node模块的本地版本的步骤，而不是Atom中的apm。我们使用 `atom-keymap` 作为一个例子：

```
$ git clone https://github.com/atom/atom-keymap.git
$ cd atom-keymap
$ npm install
$ npm link
$ apm rebuild # This is the special step, it makes the npm work with Atom's version of No
$ cd WHERE-YOU-CLONED-ATOM
$ npm link atom-keymap
$ atom # Should work!
```

然后，当你修改了node模块的代码时，你必须运行 `npm install` 和 `apm rebuild` 。

通过服务和其它包交互

Atom包可以通过叫做服务的带有版本控制的APi，和其它包进行交互。在你的`package.json`文件中指定一个或者多个版本号来提供服务，每个版本号都要带有一个包的主模块中的方法。

```
{
  "providedServices": {
    "my-service": {
      "description": "Does a useful thing",
      "versions": {
        "1.2.3": "provideMyServiceV1",
        "2.3.4": "provideMyServiceV2",
      }
    }
  }
}
```

在你的包的主模块中实现上面的方法。这些方法会在一个包被激活的任何时候调用，它们会使用它们的通信服务。它们应该返回实现了服务API的一个值。

```
module.exports =
  activate: -> # ...

  provideMyServiceV1: ->
    adaptToLegacyAPI(myService)

  provideMyServiceV2: ->
    myService
```

与之相似，指定一个或多个版本范围来使用一个服务，每个都带有一个包的主模块中的方法。

```
{
  "consumedServices": {
    "another-service": {
      "versions": {
        "^1.2.3": "consumeAnotherServiceV1",
        ">=2.3.4 <2.5": "consumeAnotherServiceV2",
      }
    }
  }
}
```

这些方法会在一个包被激活的任何时候调用，它们会提供它们的通信服务。它们会接受到一个通信对象作为一个参数。你通常需要在包提供的服务失效的时间中，进行同种类型的清除工作。从你使用服务的方法中返回一个`Disposable`来完成它：

```
{Disposable} = require 'atom'

module.exports =
  activate: -> # ...

  consumeAnotherServiceV1: (service) ->
    useService(adaptServiceFromLegacyAPI(service))
    new Disposable -> stopUsingService(service)

  consumeAnotherServiceV2: (service) ->
    useService(service)
    new Disposable -> stopUsingService(service)
```

维护你的包

虽然到目前为止，你在开发一个包的时候，发布是最通常的行为，但是你还需要做一些其它的事情。

撤销发布一个版本

如果你错误地发布了你的包的一个版本，或者你发现了一个显眼的bug或安全漏洞，你可能想要撤销这个版本的发布。例如，如果你的包叫做 `package-name` 而且错误的版本是v1.2.3，你可以执行如下命令：

```
apm unpublish package-name@1.2.3
```

这会从 https://atom.io/ 包注册处移除特定的版本。任何下载了这个版本的人会依然保留它，但是它不再对其它人提供安装。

添加协作人

一些包对于一个人来说太大了。有时优先级会更改，或者其它人想要提供帮助。你可以在你的包的GitHub仓库中，通过添加它们到协作者，来让其它人帮忙或者创建共同拥有者。注意：任何具有你的仓库推送（push）权限的人，都可以发布属于这个仓库的包的新版本。

你也可以拥有属于Github组织的包。任何人如果属于一个组织，并且这个组织具有一个包所在仓库的的推送权限，它就可以发布这个包的新版本。

转移控制权

这是一个永久的改变，而且没有办法撤销！

如果你想要把你的包的支持移交给其它人，你应该向新的拥有者转移这个包的仓库。

撤销发布你的包

在删除你的仓库之前撤销发布你的包非常重要。如果你首先删除了仓库，你就会失去对包的访问途径，并且在没有协助之下不能将其恢复。

如果你不再对你的包提供支持，并且找不到任何人来接盘，你可以从 https://atom.io/ 中撤销发布你的包。例如，如果你的包叫 `package-name` ，你可以执行如下命令：

```
apm unpublish package-name
```

这个命令会从 https://atom.io/ 包注册处移除你的包。任何下载了你的包的副本的人依然会保留，以及能够使用它，但是它不再对其它人提供安装。

重命名你的包

如果由于任何原因你需要重命名你的包，你可以使用一条简单的命令`apm publish --rename`，来修改你的包的`package.json`文件中的`name`字段，推送（push）一个新的提交（commit）和打上标签（tag），以及发布重命名之后的包。向之前名字发送的请求会重定向到新的名字。

一旦一个包的名称被使用，它就不能其它包复用，即使原来的包撤销了发布。

```
apm publish --rename new-package-name
```

小结

现在你应该对Atom核心APi和系统有了更深入的理解。